ADRIAN I. VLAD

# BASIC

# PROGRAME DE INSTRUIRE

ing. Adrian I. Vlad

PROGRAME DE INSTRUIRE

**Nota editurii**
Această lucrare a fost elaborată, tehnoredactată şi culeasă în întregime cu ajutorul calculatorului.

ing. ADRIAN I. VLAD

# PROGRAME DE INSTRUIRE

EDITURA ŞTIINŢIFICĂ ŞI ENCICLOPEDICĂ

Bucureşti, 1989

I.S.B.N. 973-29-0108-x

# CUVÎNT ÎNAINTE

*În mai puţin de cinci decenii de la apariţia lor, mijloacele tehnice moderne de calcul electronic au devenit instrumente obişnuite de lucru în cele mai diverse domenii ale activităţii umane. Aceasta a fost posibilă atît datorită creşterii cu totul deosebite a performanţelor cît şi a dezvoltării concomitente a unei diversităţi de limbaje de programare cu ajutorul cărora s-a realizat o gamă largă de programe de aplicaţii. Între aceste limbaje de programare, un loc aparte îl ocupă limbajul BASIC (Beginners All-purpose Symbolic Instruction Code).*

*Uşor de învăţat şi simplu de utilizat, acest limbaj este implementat pe aproape toate tipurile de calculatoare, permiţînd rezolvarea unor probleme dintre cele mai diverse şi complexe.*

*În lume sînt realizate, în prezent, numeroase dialecte ale limbajului BASIC. Lucrarea de faţă prezintă cea mai uzuală versiune (BASIC Microsoft), pornind de la considerentul că acest dialect este foarte larg răspîndit.*

*Limbajul BASIC este implementat pe numeroase tipuri de calculatoare produse de cele mai cunoscute firme, inclusiv pe calculatorul FELIX-PC realizat de Întreprinderea de Calculatoare Bucureşti.*

*Această carte se doreşte a fi un instrument de lucru la îndemîna utilizatorilor calculatoarelor personale. Pornind de la acest scop, se vor prezenta în cele cinci capitole ale lucrării atît informaţiile generale necesare în procesul programării cît şi instrucţiunile BASIC în ordine alfabetică. În acest fel va fi uşurată atît lectura cît şi identificarea rapidă a informaţiilor utile în timpul sesiunii de lucru la calculator.*

*Lucrarea a fost în întregime realizată pe calculator, iar exemplele prezentate au fost atent verificate şi rulate pentru a se elimina eventualele erori.*

Autorul

# CUPRINS

# 1. SESIUNEA DE LUCRU BASIC

## 1.1. Opţiuni în linia de comenzi BASIC

O sesiune de lucru BASIC este perioada de timp în care se lucrează sub acest limbaj de programare. Avînd în vedere că interpretorul sau compilatorul limbajului este rezident pe disc, este necesar ca la debutarea şesiunii de lucru să se apeleze limbajul de pe suportul magnetic în memoria calculatorului, iar la sfîrşitul sesiunii să se execute întoarcerea la sistemul de operare.

Apelarea limbajului se face din linia de comandă a sistemului de operare cu numele lui (BASIC, BASICA, GWBASIC, etc). Numele acestuia poate fi urmat de opţiunile:

*BASIC [numefis] [stdin] [ < ] [stdout] [/F:files] [/S:bsize] [C:combuffer] [/M:[max workspace] [,max blockspace]] [/D]*

unde:

**numefis** repezintă numele programului care va fi încărcat în memorie şi rulat imediat. Dacă nu se va găsi o altă extensie se va lua .BAS drept extensie de fişier, aceasta permiţînd programelor BASIC să fie rulate în modul "batch processing" prin punerea acestei linii de comandă în fişierul AUTOEXEC.BAT.

**stdin** un program BASIC are drept periferic de intrare, în mod normal, claviatura. Utilizarea opţiunii **stdin** permite limbajului să recepţioneze intrarea de la un fişier specificat.

**stdout** un program BASIC atribuie ecranul drept periferic de ieşire. Utilizînd opţiunea **stdout** se informează limbajul asupra perifericului sau fişierului de ieşire.

- **/F:files** atribuie numărul maxim de fişiere ce pot fi deschise simultan în timpul rulării unui program BASIC. Fiecare fişier necesită 62 octeţi de memorie pentru "blocul de control al fişierelor" (FCB), plus 128 octeţi pentru memoria tampon de date. Dimensiunea memoriei tampon poate fi modificată cu ajutorul opţiunii **/S:**. În cazul în care opţiunea **/F:** este omisă, numărul prestabilit de fişiere este 3.

**/S:bsize** atribuie mărimea memoriei tampon pentru fişierele de tip

"aleator". Lungimea parametrului "record" în instrucţiunea OPEN nu poate depăşi această valoare. Mărimea memoriei tampon standard este de 128 octeţi. Valoarea maximă acceptată este de 32767 octeţi.

**/C:commbuffer** atribuie mărimea tamponului de memorie pentru recepţia datelor în cazul utilizării "Adaptorului de comunicaţii asincrone" (Asynchronous Communications Adapter). Această opţiune nu are nici un efect atîta timp cît nu există ataşat sistemului acest adaptor. Memoria tampon de transmisie de date este de 128 octeţi. Valoarea maximă care se poate atribui este 32767. Dacă se omite opţiunea **/C:commbuffer**, pentru tamponul de memorie de recepţie se vor atribui 256 octeţi. În cazul în care sistemul are ataşată o linie de transmisie de mare viteză se poate utiliza opţiunea **/C:1024**.

**/M:max workspace** atribuie valoarea maximă a locaţiei care va fi utilizată de către limbaj. Limbajul BASIC atribuie 64Ko de memorie pentru date şi stivă. Dacă se intenţionează să se utilizeze subrutine în limbaj masină, se recomandă utilizarea opţiunii **/M:** în linia de comandă BASIC. De exemplu, **/M:32768** alocă 32768 octeţi pentru datele şi stiva programului BASIC şi rezervă 32768 octeţi pentru rutinele limbaj masină.

**:max blockspace** specifică numărul maxim de paragrafe alocate pentru BASIC, plus spaţiul de memorie aflat deasupra stivei şi care poate fi utilizat pentru rutinele în limbaj masină. Paragrafele sînt blocuri de cîte 16 octeţi fiecare. Dacă **:max blockspace** este omis, se atribuie valoarea &H1000 (4096) şi, deci, 65536 (4096*16) octeţi pentru zona de date şi stiva BASIC. Spre exemplificare, dacă se doresc 65535 octeţi pentru BASIC şi 512 octeţi pentru subrutinele în limbaj masină, se va utiliza **/M:,&H1020** (4096 paragrafe pentru BASIC + 32 paragrafe pentru rutinele limbaj masină).

**/D** indică limbajului că funcţiile matematice dublă precizie vor rămîne rezidente. La specificarea opţiunii **/D** aproximativ 3000 octeţi vor rămîne rezidenţi în BASIC pentru utilizarea funcţiilor transcedentale. Funcţiile care pot fi convertite în dublă precizie sînt: ATN, COS, EXP, LOG, SIN, SQR si TAN.

Cîteva exemple de linii de comandă BASIC:

**BASIC progl.bas**

încarcă şi execută fişierul **progl.bas.**

**BASIC prog2.bas/F:6**

încarcă şi execută programul **prog2.bas** utilizînd 64 Ko de memorie şi 6 fişiere.

**BASIC /C:0/M:32768**

dezafectează interfaţa RS232 şi utilizează 32 Ko pentru spaţiul de memorie de lucru BASIC.

În lucrare, la prezentarea sintaxei diferitelor instrucţiuni ori comenzi, s-au folosit următoarele convenţii:

- cuvintele scrise cu litere mari sînt cuvinte cheie şi trebuie introduse aşa cum sînt prezentate, cu precizarea că ele pot fi scrise cu litere mari sau mici sau cu orice combinaţie a lor. Limbajul BASIC face conversia literelor mici în litere mari, exceptînd cazul în care ele fac parte dintr-un şir de caractere sau o instrucţiune DATA.
- orice termeni prezentaţi mai sus cu litere mici trebuie înlocuiţi cu o valoare.
- termenii prezentaţi între paranteze drepte [ ] sînt opţionali.
- parantezele rotunde ( ) indică faptul că termenii interiori pot fi repetaţi de cîte ori este nevoie.
- toate semnele de punctuaţie cu excepţia parantezelor pătrate (virgule, paranteze, punct şi virgulă, sau semnul egal), trebuiesc incluse acolo unde sînt indicate.

## 1.2 Redirecţionarea intrării şi ieşirii standard

Limbajul BASIC permite ca intrarea şi/sau ieşirea standard să fie redirecţionată. De obicei, pentru intrare se foloseşte claviatura, iar pentru ieşire ecranul. Acestea pot fi înlocuite, după dorinţă, cu un fişier sau un alt periferic de intrare sau ieşire. Pentru aceasta sintaxa este:

**BASIC numefis[ < stdin] [ < ] [ < stdout]**

- în timpul redirecţionarii, instrucţiunile INPUT, INPUT$, INKEY$ şi LINE INPUT vor citi datele de la fişierul sau perifericul de intrare, în loc de claviatură.
- instrucţiunile PRINT şi eventualele mesaje de eroare se vor scrie pe fişierul sau perifericul de ieşire, în locul ecranului.
- cînd un fişier este redirecţionat doar pentru ieşire, toate datele ce apar pe ecran vor fi trimise la fişierul de ieşire specificat.
- cînd se redirecţionează doar ieşirea, mesajele de eroare vor merge atît la ecran cît şi la fişierul de ieşire, după care vor fi închise toate fişierele, programul se va termina, iar controlul va fi preluat de către sistemul de operare DOS.
- fişierul de intrare "KYBD:" va citi datele de la claviatură.
- fişierul de ieşire "SCRN:" va afişa datele pe ecran.
- comanda Ctrl-Break va acţiona la ieşirea standard, va închide toate fişiere şi va da controlul sistemului de operare DOS.

Cîteva exemple de redirecţionare:

**BASIC prog > date.out**

În exemplul de mai sus, ieşirea va fi redirecţionată spre fişierul **date.out**,

înlocuindu-se astfel ecranul.

**BASICA unprog < date.in**

În acest exemplu, instrucţiunile de intrare (INPUT, INPUT$, LINE INPUT şi INKEY$) vor fi primite de la fişierul **date.in**, înlocuindu-se claviatura.

**BASIC unprog < date.in > prog.fin**

Datele citite cu instrucţiunile INPUT, INPUT$, INKEY$ sau LINE INPUT vor fi recepţionate de la fişierul **date.in**, iar datele scrise cu instrucţiunea PRINT vor fi adăugate fişierului **prog.fin**.

# 2. MODURI DE OPERARE

În momentul inițializării, limbajul BASIC va afișa pe ecran mesajul **Ok**, indicînd faptul că este gata pentru recepționarea instrucțiunilor. Această stare se numește nivel de comandă. Comunicația cu BASIC-ul se poate realiza în două moduri: modul direct și modul program.

## 2.1. Modul direct

În modul direct, se poate instrui limbajul să execute o comandă imediat după ce a fost introdusă. Aceasta se realizează prin omiterea numărului de linie. Astfel, se pot afișa rezultatele unor operații aritmetice sau logice sau se pot memora pentru utilizarea lor ulterioară, singura remarcă fiind că instrucțiunile respective nu sînt memorate după afișarea rezultatului. Acest mod este avantajos pentru depanarea programelor, cît și pentru unele calcule rapide, care nu necesită realizarea unui program. De exemplu:

```
PRINT 10*4.52
45.2
```

## 2.2. Modul program

În modul program (sau modul indirect), se instruiește limbajul BASIC că liniile introduse alcătuiesc un program. Pentru aceasta se va începe o linie cu un număr de linie. Astfel, această linie va fi memorată ca parte a unui program în memorie. Programul astfel memorat poate fi rulat cu comanda RUN. De exemplu:

```
10 PRINT 10*4.52
RUN
45.2
```

# 3. INFORMAŢII GENERALE DESPRE PROGRAMAREA ÎN BASIC

## 3.1. Formatul liniei

O linie BASIC are următorul format general:

*nnnnn instrucţiune BASIC[: instrucţiune BASIC...][' comentariu]*

şi se termină cu caracterul ENTER.

**nnnnn** reprezintă numărul liniei de program BASIC şi este un număr de 1 pînă la 5 cifre lungime. Acesta indică ordinea în care au fost memorate liniile de program BASIC şi serveşte, de asemenea, drept referinţa pentru salturile de program precum şi pentru editare. Numărul liniei poate fi cuprins între 0 şi 65529.

**instrucţiune BASIC** O instrucţiune BASIC poate fi executabilă sau neexecutabilă. Instrucţiunile executabile indică limbajului ce acţiune trebuie îndeplinită în momentul rulării. De exemplu instrucţiunea PRINT este o instrucţiune executabilă. Instrucţiunile neexecutabile ca de exemplu DATA sau REM, conţin doar informaţii şi nu pot cauza o anumită acţiune directă în timpul rulării. Toate instrucţiunile BASIC vor fi descrise în capitolele viitoare.

După dorinţă, o linie poate include mai mult de o instrucţiune, dar fiecare instrucţiune trebuie separată de celelalte prin două puncte (:), cu menţiunea că o linie de program nu poate avea mai mult de 255 de caractere inclusiv numărul liniei. De exemplu:

```
10 FOR I = 1 TO 5: PRINT I: NEXT I
RUN
1
2
3
4
5
```

comentariu Comentariul poate fi inclus la sfîrşitul unei linii. Simbolul ' separă comentariul de restul liniei.

### 3.2. Constante

Constantele sînt valori ce se utilizează în timpul elaborării unui program, dar care nu se schimbă în timpul execuţiei unui program. Acestea pot fi: constante şir (de caractere) şi constante numerice.

O constanta şir este o secvenţă de pînă la 255 de caractere incluse între ghilimele ("). Caracterele pot fi litere, cifre sau simboluri. Exemple de constante şir de caractere:

**"Abcd"**
**"a12.123"**
**"1"**
**"2#$%&"**

Constantele numerice sînt numere pozitive sau negative. Constantele numerice în BASIC nu pot conţine virgula zecimală drept delimitator ci (după notaţia engleză) punctul. Există cinci moduri de reprezentare a constantelor numerice:

**întregi** toate numerele între -32768 şi +32767. Constantele întregi nu pot conţine punctul zecimal.

**virgulă fixă** numerele reale pozitive sau negative.

**virgulă mobilă** numerele pozitive sau negative reprezentate în forma exponenţială (similară notaţiei ştiinţifice). În calculele în simplă precizie o constantă în virgulă mobilă constă dintr-un întreg pozitiv sau negativ sau un număr în virgulă fixă (mantissa) urmată de litera E, urmată de un întreg pozitiv sau negativ (exponentul). Constantele de tip dublă precizie utilizează litera D în locul lui E. E sau D înseamnă "ori zece la puterea". Se poate reprezenta, în acest mod, orice număr pozitiv sau negativ cuprins între 2.9E-39 la 1.7E+38.

### 3.3. Precizia numerică

Intern, numerele pot fi memorate ca numere întregi, simplă precizie sau dublă precizie. Constantele declarate în format întreg, hexazecimal sau octal sînt memorate în doi octeţi şi sînt interpretate ca numere întregi. În calculele de dublă precizie, numerele sînt memorate cu 17 cifre precizie şi sînt afişate cu pînă la 16. În calculele de simplă precizie se memorează 7 cifre şi se afişează pînă la 7 cifre, din care doar 6 pot fi de această precizie. Exemple de constante de simplă şi dublă precizie:

| **simplă precizie** | **dublă precizie** |
|---|---|
| **46.8** | **235468274** |
| **-1.09E-6** | **-1.09432D-6** |
| **3489.0** | **11.22334567871346** |

## 3.4. Variabile

Variabilele sînt nume utilizate pentru a reprezenta valori în programele BASIC. Ca şi în cazul constantelor, există două tipuri de variabile: numerice şi şir (de caractere). O variabilă numerică are totdeauna ataşată o valoare numerică. Unei variabile şir i se atribuie drept valoare un şir de caractere. Şirul poate avea pînă la 255 de caractere.

Unei variabile i se poate atribui o valoare fixă sau rezultată în urma unor calcule. În orice caz tipul de variabilă trebuie să concorde cu datele care i se atribuie.

Dacă se utilizează o variabilă înainte de a i se atribui o valoare, acesteia i se atribuie, din oficiu, valoarea zero pentru variabilele numerice şi valoarea nulă (şir de lungime zero) pentru variabilele şir.

BASIC-ul permite nume de variabile de pînă la 40 de caractere lungime. Caracterele pot fi litere, cifre sau punctul zecimal, cu condiţia ca primul caracter al numelui variabilei să fie o literă. Caracterele speciale care identifică tipul variabilei declarate trebuie să se afle pe ultima poziţie a numelui variabilei. O variabilă nu poate avea un nume rezervat de instrucţiunile BASIC. Astfel instrucţiunea:

**10 COS = 1**

nu este corectă deoarece COS este un nume rezervat limbajului şi nu poate fi, deci, numele unei variabile. În schimb:

**10 COSINUS = 1**

este o instrucţiune corectă. De notat faptul că o variabilă care începe cu prefixul FN se aşteaptă a fi o variabilă declarată funcţie, definită de utilizator (vezi declaraţia DEF FN).

## 3.5. Conversia numerelor dintr-o precizie în alta

Un nume de variabilă determină tipul (şir de caractere, numerică) sau în cazul variabilelor numerice, precizia. După cum am spus, un caracter special aflat la sfîrşitul numelui unei variabile indică tipul acesteia. Astfel, caracterul:

- declară o variabilă numerică simplă precizie **A = 12.345**

**!** - declară o variabilă numerică simplă precizie **A! = 11.23412**

**%** - declară o variabilă numerică întregă **A% = 123**

**#** - declară o variabilă numerică dublă precizie **A# = 1.123472238123**

**$** - declară variabilă şir de caractere **A$ = "ABCD"**

Tipul variabilei numerice poate fi declarat şi pe altă cale. Astfel, instrucţiunile DEFINT, DEFSNG, DEFDBL, DEFSTR pot fi şi ele utilizate

în program pentru a declara tipul de variabilă (vezi instrucţiunile DEFtip).

Cînd este necesar, limbajul BASIC converteşte numerele dintr-o precizie în alta după următoarele reguli:

1. Dacă o valoare numerică de o precizie este atribuită unei variabile numerice de precizie diferită, numărul este memorat cu precizia variabilei ţintă. Exemplu:

```
10 A% = 4.3256
20 PRINT A%
RUN
4
```

2. Dacă se atribuie unei variabile de precizie mai mică o valoare numerică de precizie mai mare se execută o rotunjire şi nu o trunchiere:

```
10 B = 7.394920883132416
20 PRINT B
RUN
7.394921
```

Această regulă afectează nu numai instrucţiunile de atribuire (de ex. I% = 2.5 implică I% = 3), dar, de asemenea, funcţiile şi instrucţiunile de evaluare. De exemplu TAB(4.5) va executa TAB(5); A(2.5) va avea drept rezultat A(3), iar X = 11.5 MOD 4 va fi evaluat X = 0.

3. Dacă se face conversia dintr-o precizie mai scăzută, într-o precizie mai ridicată numărul rezultat nu poate avea valoare mai precisă. Exemplu:

```
10 A = 2.04
20 B# = A
30 PRINT A;B#
RUN
2.04   2.039999961853027
```

Eroarea poate fi limitată utilizînd următoarea formulă:

```
ABS(B#-A) < 6.3E-8 * A
```

4. Cînd este evaluată o expresie, toţi operanzii unei operaţii aritmetice sau relaţionale sînt convertiţi la acelaşi grad de precizie, adică la precizia maximă. De asemenea, rezultatul unei operaţii aritmetice va fi afişat în această precizie. De exemplu:

```
10 D# = 6#/7
20 PRINT D#
RUN
.8571428571428571
```

Operaţia aritmetică se face în dublă precizie, iar rezultatul este atribuit variabilei D# ca o valoare în dublă precizie.

```
10 D = 6#/7
20 PRINT D
```

```
RUN
.857429
```

Operaţia aritmetică se face în dublă precizie, iar rezultatul se atribuie variabilei D ca o valoare în simplă precizie.

5. Opertorii logici convertesc operanzii în întregi, iar rezultatul va fi un număr întreg. Operanzii trebuie să fie cuprinşi între -32768 şi 32767, în caz contrar se va semnala eroare de depăşire (**Overflow**).

## 3.6. Matrici

O matrice (sau vector) este o listă sau un tablou de valori la care se pot face referiri cu un singur nume. Fiecare valoare dintr-o matrice se numeşte element. Elementele sînt şiruri sau valori numerice şi pot fi utilizate în expresii sau în instrucţiuni BASIC.

Subdomeniul (numărul din paranteză) indică poziţia elementului în matrice. Prima poziţie este zero atîta timp cît nu este schimbată cu ajutorul instrucţiunii OPTION BASE. Declararea numelui şi tipului unei matrici precum şi a numărului de elemente se numeşte definirea sau dimensionarea matricei. Numărul maxim de dimensiuni ale unei matrici este 255. Se poate utiliza o matrice fără a o dimensiona, în acest caz subdomeniul maxim este de 10 (deci 11 elemente numerotate de la 0 la 10). Pentru definirea unei matrici se utilizează instrucţiunea DIM. De exemplu:

**DIM B$(5)**

va crea o matrice de tip şir unidimensională cu numele B$, cu 6 elemente maxim:

B$(0)
B$(1)
B$(2)
B$(3)
B$(4)
B$(5)

Un alt exemplu:

**DIM A(2,3)**

Această instrucţiune crează o variabilă numerică A, bidimensională. Atîta timp cît matricea nu include un caracter special de declarare de tip de variabilă, aceasta se consideră a fi simplă precizie.

Matricea A poate fi gîndită ca un tablou de 3 rînduri şi patru coloane:

| | | | |
|---|---|---|---|
| A(0,0) | A(0,1) | A(0,2) | A(0,3) |
| A(1,0) | A(1,1) | A(1,2) | A(1,3) |
| A(2,0) | A(2,1) | A(2,2) | A(2,3) |

O variabilă scalară poate avea acelaşi nume cu o matrice deoarece

acestea sînt reprezentate intern diferit. Astfel:

**A = 7** si **A(3,3)**

nu se confundă deoarece A este numele unei variabile scalare, iar A(...) este numele unui vector.

### 3.7. Expresii numerice şi operatori

O expresie numerică poate fi o variabilă sau o constantă numerică. De asemenea, poate fi un operator, combinînd constante şi variabile pentru a produce o singură valoare numerică.

Operatorii numerici realizează operaţii numerice sau logice de cele mai multe ori asupra valorilor numerice, dar şi asupra valorilor şir de caractere. Ei sînt numiţi operatori numerici deoarece au drept rezultat o valoare numerică, adică un număr. Operatorii numerici pot fi divizaţi în următoarele categorii:

- aritmetici
- relaţionali
- logici
- funcţii

### 3.8. Operatori aritmetici

Operatorii aritmetici realizează operaţiile aritmetice uzuale în ordinea matematică standard:

| Operator | Operaţie | Exemplu |
|---|---|---|
| ^ | Exponenţial | X ^ Y |
| - | Negaţie | -X |
| *,/ | Multiplicare | X*Y |
| | Diviziune în virgulă mobilă | X/Y |
| \ | Diviziune întreagă | X\Y |
| MOD | Modulo aritmetic | X MOD Y |
| +,- | Adunare | X + Y |
| | Scădere | X-Y |

Cele mai multe dintre aceste operaţii fiind, probabil, cunoscute de către cititor, ne vom opri doar la operaţiile de diviziune întreagă şi modulo aritmetic.

**Diviziunea întreagă** se notează cu simbolul (\). Operanzii sînt rotunjiţi la întregi cu valori între -32768 şi 32767 înainte de a fi realizată

diviziunea, cîtul fiind trunchiat la un întreg. De exemplu:

```
10 A = 10\4
20 B = 25.68\6.99
30 PRINT A;B
RUN
2  3
```

**Modulo aritmetic** se notează cu operatorul **MOD**. El dă valoarea întreagă a restului unei împărţiri întregi. De exemplu:

```
PRINT 7 MOD 4
3
```

deoarece 7/4 = 1 rest 3.

```
PRINT 25.68 MOD 6.99
5
```

rezultă 5 deoarece 26/7 = 3 rest 5.

### 3.9. Operatori relaţionali

Operatorii relaţionali compară două valori. Valorile pot fi atît de tip şir cît şi numerice. Rezultatul comparării va fi *adevărat* (-1) sau *fals* (0).

| Operator | Relaţia testată | Exemplu |
|---|---|---|
| = | Egalitate | X = Y |
| < > sau > < | Inegalitate | X< >Y |
| < | Mai mic ca | X<Y |
| | Mai mare ca | X>Y |
| < = sau = < | Mai mic sau egal ca | X< =Y |
| = sau = > | Mai mare sau egal ca | X > = Y |

**Comparări de valori numerice**. Cînd operatorii aritmetici şi relaţionali se combină într-o expresie, mai întîi se execută operaţiile aritmetice. De exemplu expresia:

**X+Y < (A-1)/B**

este *adevărată* (-1) dacă valoarea X adunată cu valoarea Y este mai mică decît valoarea A-1 împărţită la B.

**Comparări de şiruri**". Comparările de şiruri pot fi gîndite ca "alfabetice". Astfel un şir este *mai mic ca* un altul dacă o literă de pe o poziţie a primului şir vine alfabetic după o literă aflată pe o poziţie echivalentă în şirul al doilea. Literele mici sînt *mai mari decît* aceleaşi litere mari. De exemplu:

**"AA" < "AB"**
**"kg" < "KG"**

"123" < "678"
"abc" > "123"

### 3.10. Operatori logici

Operatorii logici execută operaţii logice (sau booleene) asupra valorilor numerice. Un operator logic combină valori adevărate - false şi atribuie un rezultat la rîndul lui adevărat sau fals. Un operand este considerat a fi *adevărat* dacă este diferit de zero şi *fals* dacă este egal cu zero. Asupra numerelor operaţia se efectuează bit cu bit. Operatorii logici sînt:

| | |
|---|---|
| **NOT** | complement logic |
| **AND** | conjuncţie |
| **OR** | disjuncţie |
| **XOR** | sau exclusiv |
| **EQV** | echivalenţă |
| **IMP** | implicaţie |

Fiecare operator atribuie rezultatul după cum este ilustrat în tabelul alăturat (unde A - *adevărat*, F - *fals*). Operatorii sînt listaţi în ordinea precedenţei:

| | | | |
|---|---|---|---|
| NOT | X | | NOT X |
| | A | | F |
| | F | | A |
| AND | X | Y | X AND Y |
| | A | A | A |
| | A | F | F |
| | F | A | F |
| | F | F | F |
| OR | X | Y | X OR Y |
| | A | A | A |
| | A | F | A |
| | F | A | A |
| | F | F | F |
| XOR | X | Y | X XOR Y |
| | A | A | F |
| | A | F | A |
| | F | A | A |
| | F | F | F |

| | X | Y | X EQV Y |
|---|---|---|---|
| EQV | X | Y | X EQV Y |
| | A | A | A |
| | A | F | F |
| | F | A | F |
| | F | F | A |
| IMP | X | Y | X IMP Y |
| | A | A | A |
| | A | F | F |
| | F | A | A |
| | F | F | A |

Operatorii sînt convertiţi la întregi între -32768 şi 32767. Dacă valoarea operanzilor este în afara acestui domeniu se semnalează eroare de depăşire (**Overflow**). Dacă operandul este negativ se foloseşte forma complementului faţă de doi. Aceasta transformă operandul în secvenţe de 16 biţi. Astfel operaţia se execută asupra acestor secvenţe. Următorul exemplu ilustrează modul în care lucrează operatorii logici:

**A = 63 AND 16**

În acest caz A ia valoarea 16. Deoarece 63 în binar este 111111, iar 16 este 10000, rezultă că 63 AND 16 este egal cu 010000, care este egal cu 16 în decimal.

**B = -1 AND 8**

B va lua valoarea 8 deoarece -1 în binar este 1111111111111111 şi 8 în binar este 1000, -1 AND 8 va fi egal cu 0000000000001000, sau 8 în zecimal.

**C = 4 OR 2**

C va fi egal cu 6 deoarece 4 în binar este 100 şi 2 este 010, 4 OR 2 va fi în binar 110, deci 6.

### 3.11. Funcţii numerice

O funcţie este utilizată ca o variabilă într-o expresie pentru a chema o operaţie predeterminată ce urmează a fi executată pe unul sau mai mulţi operanzi. Limbajul BASIC are cîteva funcţii numerice încorporate ce rezidă în sistem, ca de exemplu SQR (rădăcina pătrată), SIN (sinus), etc. De asemenea utilizatorul poate defini propriile funcţii cu ajutorul instrucţiunii DEF FN.

Anterior, am discutat operaţiile numerice în ordinea lor de precedenţa. Pe scurt:

1. Funcţia chematoare este evaluată prima.
2. Operaţiile aritmetice sînt executate în ordinea:

   ^ (ridicarea la putere)

   \- (scăderea)

*,/ (înmulţirea şi împărţirea)
\ (împărţirea întreagă)
MOD (modulo aritmetic)
+,- (adunarea şi scăderea)
3. Operaţiile relaţionale
4. Operaţiile logice sînt executate ultimele în ordinea: NOT, AND, OR, XOR, EQV SI IMP.

Operaţiile cu acelaşi nivel de execuţie sînt realizate în ordinea "de la stînga la dreapta". Pentru a schimba ordinea de execuţie a operaţiilor se vor folosi parantezele. În interiorul parantezelor nivelul uzual al ordinii de execuţie se păstrează.

### 3.12. Expresii şi operatori şir

O expresie şir poate fi o constantă şir, o variabilă şir, sau o combinaţie a acestora prin utilizarea operatorilor pentru a produce o singură valoare şir. Operatorii şir sînt folosiţi pentru aranjarea şirurilor de caractere în diferite moduri. Cele două categorii de operatori şir sînt:

- concatenarea
- funcţia

De notat că se pot folosi operatorii relaţionali (=,< >,<,>,< =,> =) pentru compararea a două şiruri, însă aceştia nu sînt consideraţi "operatori şir" deoarece au ca rezultat o valoare numerică şi nu o valoare şir.

Operaţia de legare a două şiruri de caractere se numeşte concatenare. Această operaţie se realizează cu ajutorul simbolului ( + ). De exemplu:

```
10 A$="Calculatorul"
20 B$=" Felix PC"
30 C$=A$+B$
40 PRINT C$
RUN
Calculatorul Felix PC
```

Funcţia şir este identică cu funcţia numerică cu deosebirea că rezultatul primeia este o valoare numerică. O funcţie şir poate fi folosită într-o expresie pentru chemarea unei operaţii predeterminate ce urmează a fi executată pe unul sau mai mulţi operanzi. Limbajul BASIC are înglobate mai multe funcţii şir ce rezidă în sistem, acestea urmînd a fi explicate şi detaliate în unul din capitolele viitoare.

# 4. INTRĂRI ŞI IEŞIRI

## 4.1. Fişiere

Un fişier este o colecţie de informaţii ce este "păstrat" pe un suport, altul decît RAM (Random Access Memory), de exemplu disc, bandă magnetică, etc. Pentru a avea acces la informaţiile fişierului este necesară deschiderea acestuia cu ajutorul instrucţiunii OPEN. După aceasta fişierul poate fi utilizat pentru operaţii de intrare-ieşire. Limbajul suportă conceptul de **fişiere generale Input/Output**. Aceasta înseamnă că orice tip de intrare/ieşire (I/O) poate fi tratat ca I/O către un fişier, indiferent dacă se utilizează un disc, o bandă, sau un canal de comunicaţie cu un alt calculator.

BASIC-ul execută operaţiile de I/O utilizînd un număr de fişier. În momentul deschiderii unui fişier cu ajutorul instrucţiunii OPEN se atribuie un număr de fişier sau periferic. Numărul de fişier poate fi orice număr, variabilă sau expresie în domeniul de la 1 la numărul maxim de fişiere ce pot fi deschise simultan.

## 4.2. Numele de fişier

Acesta trebuie să fie în conformitate cu convenţiile DOS (Disk Operating System), astfel:

- numele poate consta din două părţi distincte separate prin punct (.):

  **nume.extensie** .

Numele poate fi unul pînă la opt caractere lungime, iar extensia de unul la trei caractere. Dacă se folosesc mai mult de trei caractere pentru extensie restul vor fi ignorate. Dacă sînt folosite mai mult de opt caractere pentru nume, iar extensia nu este inclusă, limbajul va include după opt caractere un punct, iar restul de caractere (pînă la trei) vor fi folosite pentru extensie. Dacă numele este mai mare de opt caractere şi extensia prezentă se va semnala eroare.

- următoarele caractere sînt permise în numele şi extensia fişierului:
  A la Z, 0 la 9, ( ) { } @ # $ % ^ & * ! -_/"

### 4.3. Numele perifericului

Constă din pînă la patru caractere alfanumerice urmate de două puncte (:). Este numele unui periferic de intrare/ieşire:

| | |
|---|---|
| KYBD: | Claviatura. Doar pentru intrare. |
| SCRN: | Ecranul. Doar pentru ieşire. |
| LPT1: | Primul printer. Ieşire sau acces aleator. |
| LPT2: | Al doilea printer. Ieşire sau acces aleator. |
| LPT3: | Al treilea printer. Ieşire sau acces aleator. |

PERIFERICE DE COMUNICAŢIE

| | |
|---|---|
| COM1: | Primul adaptor de comunicaţie asincronă. Intrare şi ieşire. |
| COM2: | Al doilea adaptor de comunicaţie asincronă. Intrare şi ieşire. |

PERIFERICE DE MEMORARE

| | |
|---|---|
| A: | Primul cititor de disc. Intrare şi ieşire. |
| B: | Al doilea cititor de disc. Intrare şi ieşire. |
| C: | Primul disc fix. Intrare şi ieşire. |
| D: | Al doilea disc fix. Intrare şi ieşire. |

# 5. COMENZI, INSTRUCŢIUNI ŞI FUNCŢII BASIC

În acest capitol vor fi descrise complet comenzile, instrucţiunile şi funcţiile limbajului BASIC. Acestea vor fi prezentate în ordine alfabetică. Distincţia dintre comenzile şi instrucţiunile BASIC este o problemă care ţine mai mult de tradiţie. Comenzile, deoarece, în mod general, operează asupra programelor, pot fi introduse în modul direct. Spre deosebire de acestea, instrucţiunile urmează un algoritm în cadrul unui program, de aceea ele pot fi introduse în modul indirect, deci ca parte a unui program. În dialectele BASIC moderne (ca şi cel de faţă) cele mai multe dintre comenzile şi instrucţiunile limbajului pot fi introduse atît în modul direct cît şi în modul program.

Descrierea fiecărei comenzi, instrucţiuni, funcţii sau variabile se va face după următorul algoritm:

**Scopul:** Indică utilitatea comenzii, instrucţiunii funcţiei sau variabilei.

**Formatul:** Prezintă formatul corect de scriere a comenzii, ins ıruc-ţiunii, sau a variabilei. De reţinut următoarele reguli generale:

- cuvintele scrise cu majuscule sînt cuvinte cheie şi trebuie introduse exact cum sînt prezentate, cu excepţia că ele pot fi introduse cu minuscule, majuscule sau orice combinaţie a acestora, limbajul făcînd automat conversia în majuscule.
- se pot înlocui orice termeni prezentaţi în caractere minuscule.
- termenii în paranteze pătrate ([ ]) sînt opţionali.
- parantezele rotunde (...) indică faptul că termenii din interior pot fi repetaţi de cîte ori se doreşte.
- toate semnele de punctuaţie - cu excepţia parantezelor pătrate (ca de exemplu, virgule, paranteze, punct şi virgulă, semnul egal, semnul negativ) trebuie introduse unde sînt arătate.

**Comentariu:** Descrie în detaliu modul de utilizare a instrucţiunii, comenzii, funcţiei sau variabilei.

**Exemple:** Prezintă instrucţiunile în mod direct, exemple de programe sau segmente de programe ce ilustrează modul de utilizare a co-

menzilor, instrucţiunilor, funcţiilor şi variabilelor.

## ABS

Funcţie

**Scop:** Atribuie unei variabile valoarea absolută a unei expresii.

**Format:** *v = ABS(x)*

**Comentariu:** *x* poate fi orice expresie numerică. Rezultatul este, totdeauna, un număr pozitiv sau zero.

**Exemplu:**

```
PRINT ABS(3*(-4))
12
```

## ASC

Funcţie

**Scop:** Atribuie unei variabile codul ASCII al primului caracter al şirului **x$**.

**Format:** *v = ASC(x$)*

**Comentariu:** *x$* poate fi orice expresie şir. Rezultatul funcţiei **ASC** este o valoare numerică ce reprezintă codul ASCII al primului caracter din şirul **x$**. Dacă **x$** este un şir nul se va semnala eroare **Illegal function call.**

**Exemplu:**

```
10 X$ = "ALFA"
20 PRINT ASC(X$)
RUN
65
```

## ATN

Funcţie

**Scop:** Calculează arctangentă de **x**.

**Format:** *v = ATN(x)*

**Comentariu:** *x* poate fi orice expresie. Funcţia **ATN** calculează unghiul a cărei tangentă este **x**. Rezultatul este o valoare în radiani în domeniul -PI/2 la PI/2, unde PI = 3.141593... Pentru conversia din radiani în grade hexazecimale se înmulţeşte cu 180/PI.

Valoarea poate fi calculată în dublă precizie specificînd argumentul **/D** în linia de comandă BASIC la iniţializarea limbajului

**Exemplu:**

```
PRINT ATN(3)
1.249046
```

Al doilea exemplu calculează unghiul a cărei tangentă este 1. Valoarea calculată este .7853983 radiani sau 45 grade.

```
10 PI=3.141593
20 RAD=ATN(1)
30 GRAD=RAD*180/PI
40 PRINT RAD,GRAD
RUN
.7853983        45
```

## AUTO

Comandă

**Scop:** Generează în mod automat numere de linie de program.

**Format:** *AUTO[număr][,[increment]*

**Comentariu:** *număr* este numărul de start al liniei. Se poate folosi un punct (.) în locul numărului de linie pentru a indica linia curentă.

*increment* este valoarea adăugată fiecărei linii pentru a da numărul liniei următoare.

Numărătoarea începe cu numărul indicat şi creşte, apoi, cu valoarea incrementului. Dacă se omit ambele valori, număratoarea va începe de la linia 10 şi va avea increment 10. Dacă este indicat un număr de linie urmat de virgulă, însă incrementul nu este specificat, se va atribui incrementul specificat la ultima comandă **AUTO**. Dacă numărul este omis, însă incrementul este specificat, numărătoarea liniilor va începe de la 0. Comanda **AUTO** este utilizată la numerotarea liniilor, ceea ce salvează timpul de scriere manuală a liniilor. Dacă **AUTO** generează un număr de linie deja existent în program se va tipări un asterisc (*) după numărul de linie, pentru a semnala faptul că linia respectivă se va suprapune peste una deja existentă. Astfel, în cazul în care se apasă tasta Enter imediat după asterisc, linia marcată va rămîne nemodificată, iar **AUTO** va genera următoarea linie de program. **AUTO** se încheie cu Ctrl-Break. Această ultimă linie nu va fi salvată.

**Exemplu:**

**AUTO**

va genera, în ordine, liniile 10, 20, 30, 40,...

**AUTO 100,50**

va genera, în ordine, liniile 100, 150, 200,...

**AUTO ,30**

va genera liniile 0, 30, 60, 90,...

## BEEP

Instrucţiune

**Scop:** Produce un sunet în difuzor.

**Format:** *BEEP*

**Comentariu:** Instrucţiunea BEEP produce în difuzor un sunet de 800 Hz timp de 1/4 secunde. Instrucţiunea **BEEP** are acelaşi efect cu :

**PRINT CHR$(7);**

## BLOAD

Comandă

**Scop:** Încarcă un fişier imagine de memorie în spaţiul indicat de memorie.

**Format:** *BLOAD numefis[,offset]*

**Comentariu:** *numefis* este o expresie şir care indică numele fişierului.

*offset* este o expresie întreagă între 0 şi 65535. Indică offset- ul la care fişierul va fi încărcat în segmentul curent de memorie, specificat, anterior, cu instrucţiunea DEF SEG.

Dacă offset-ul este omis, se va atribui ofsset-ul specificat la comanda BSAVE, aceasta însemnînd că fişierul va fi încărcat în memorie la aceeaşi locaţie la care a fost salvat cu ajutoul comenzii BSAVE. Aceste două comenzi sînt utile în cazul manipulării programelor în limbaj masină.

**Observaţie:** BASIC-ul nu verifică offset-ul segmentului curent de memorie la care se va încărca fişierul. Aceasta înseamnă că încărcarea se va putea face în orice zonă de memorie, inclusiv în zona stack-ului limbajului BASIC, a zonei de variabile sau a celei de program, ceea ce poate duce la serioase probleme în timpul rulării.

**Exemplu:** Exemplul următor încarcă tamponul de memorie de ecran a adaptorului de grafică color, care se găseşte la adresa &HB800. Linia 40 încarcă fişierul POZA la ofsset-ul 0, segmentul &HB800:

```
10 ' incarca segmentul de memorie de ecran
20 DEF SEG = &HB800
30 ' incarca POZA in memoria tampon de ecran
40 BLOAD "POZA",0
```

Exemplul de la comanda BSAVE ilustrează modul în care a fost salvat fişierul "POZA".

## BSAVE

Comandă

**Scop:** Salvează porţiuni din memoria calculatorului pe un pe-

periferic specificat.

**Format:** *BASVE numefis,offset,lungime*

**Comentariu:** *numefis* este o expresie şir care indică numele fişierului.

*offset* este o expresie întreagă în domeniul 0 la 65535. Acesta este offset-ul segmentului declarat anterior prin DEF SEG. Salvarea începe de la această locaţie.

*lungime* este o expresie întreagă cuprinsă între 1 şi 65535. Aceasta este lungimea imaginii de memorie care urmează a fi salvată.

Dacă se omit *lungimea* şi *offset*-ul, se va semnala eroare, iar salvarea nu se mai execută.

**Exemplu:** În acest exemplu vom vedea modul în care este salvată imaginea de memorie a ecranului pentru adaptorul de grafică color începînd cu locaţia de memorie &HB800. Este necesară utilizarea instrucţiunii DEF SEG pentru a indica adresa de start a tamponului de memorie a ecranului (&HB800). Offset-ul 0 şi lungimea &H4000 specifică faptul că tot tamponul de memorie de 16Ko al ecranului urmează să fie salvat.

```
10 DEF SEG = &HB800
20 BSAVE "POZA",0,&H4000
```

## CALL

Instrucţiune

**Scop:** Cheamă subrutine în limbaj masină.

**Format:** *CALL varnum[(variabila [,variabila]...)]*

**Comentariu:** *varnum* este numele unei variabile numerice. Valoarea variabilei indică offset-ul subrutinei în segmentul de memorie curent, aşa cum a fost definit în ultima instrucţiune DEF SEG.

*variabila* este numele unei variabile ce va fi trecut drept argument subrutinei în limbaj masină.

Instrucţiunea **CALL** este unul din modurile în care limbajul BASIC se poate interfaţa cu limbajul masină.

**Exemplu:** Linia 10 din program atribuie valoarea segmentului BASIC. Linia 30 declară toate valorile scalare şi matricile utilizate în program. Linia 40 calculează ofset-ul variabilei ARRAY în segmentul de date BASIC. Linia 50 încarcă fişierul într-o matrice întreagă, iar linia 60 cheamă rutina. Variabilele Q, B$ şi C sînt trecute ca argumente ale rutinei în limbaj masină.

```
10 DEF SEG: OPTION BASE 1
20 DEFINT A-Z
30 DIM ARRAY(512)
```

```
35 P=0: Q=5: C=0: B$="TRUE"
40 P=VARPTR(ARRAY(1))
50 BLOAD "ASM.FIL",P
60 CALL P(Q,B$,C)
```

## CDBL

Funcţie

**Scop:** Converteşte argumentul x într-un număr în dublă precizie.

**Format:** *v=CDBL(x)*

**Comentariu:** *x* poate fi orice expresie numerică.

**Exemplu:**

```
10 A=234.56
20 PRINT A;CDBL(A)
RUN
234.56 234.559982910156
```

## CHAIN

Instrucţiune

**Scop:** Transferă controlul unui alt program şi trece variabilele din programul curent către acesta.

**Format:** *CHAIN [MERGE]numefis[,[linie]][,[ALL][,DELETE domeniu]]]*

**Comentariu:** *MERGE* aduce un fişier ASCII în programul BASIC curent ca "overlay". Dacă urmează a fi unit prin opţiunea MERGE, programul de legătură trebuie să fie un fişier de tip ASCII. Exemplu:

**CHAIN MERGE "A:OVERLAY",1000**

*numefis* este o expresie şir care conţine numele fişierului. Numele fişierului este numele programului către care se va transfera controlul. Exemplu:

**CHAIN "A:PROG1"**

*linie* este un număr de linie sau o expresie care evaluează un număr de linie în programul de legătură, Specifică numărul liniei la care se va da controlul execuţiei. Dacă este omis, controlul se da la prima linie de program. Exemplu:

**CHAIN "A:PROG1",1000**

*ALL* specifică faptul că toate variabilele din programul curent vor fi trecute programului de legătură. Dacă opţiunea **ALL** este omisă, este necesară includerea în programul curent al unei instrucţiuni COMMON pentru trecerea variabilelor dorite programului de legatură. Exemplu:

**CHAIN "A:PROG1",1000,ALL**

*DELETE* lucrează identic cu instrucţiunea DELETE. Exemplu:

**CHAIN MERGE "A:OVERLAY1",1000,DELETE 1500-5000**

**Observaţii:**

1. Instrucţiunea CHAIN lasă fişierele deschise.
2. Instrucţiunea CHAIN conservă instrucţiunea OPTION BASE.
3. Fără argumentul MERGE, instrucţiunea CHAIN nu conservă tipurile de variabilă sau funcţiile definite de utilizator pentru a fi trecute programului de legătură. Astfel, oricare din instrucţiunile DEFINT, DEFSNG, DEFDBL, DEFSTR sau DEF FN, conţinînd variabile active trebuie redeclarate în programul de legătură.
4. Instrucţiunea CHAIN execută un RESTORE înainte de a da controlul programului de legătură.

## CHDIR

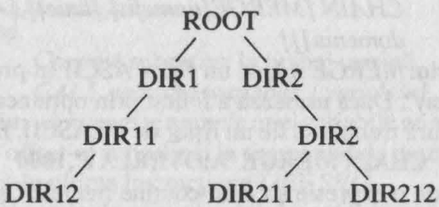
**Comandă**

**Scop:** Schimbă directorul curent.

**Format:** *CHDIR cale*

**Comentariu:** *cale* este o expresie şir, mai mică de 63 caractere, cu scopul de a identifica noul director care va deveni directorul curent.

**Exemplu:**




Pentru trecerea din directorul rădăcină (ROOT) în oricare din subdirectoare se utilizează:

**CHDIR "\"**

Pentru trecerea din directorul rădăcină în directorul DIR12 se utilizează:

**CHDIR "DIR1\DIR11\DIR12"**

Pentru trecerea, de exemplu, din directorul DIR21 în directorul DIR212 se va utiliza:

**CHDIR "DIR212"**

Pentru trecerea înapoi din directorul DIR11 în directorul DIR1, se va utiliza:

**CHDIR ".."**

## CHR$

Funcţie

**Scop:** Converteşte un cod ASCII în caracterul său echivalent.

**Format:** *v$ = CHR$(n)*

**Comentariu:** n trebuie să fie în domeniul 0 la 255. CHR$ este utilizat de obicei pentru a trimite un caracter special la imprimantă sau ecran. Complementara ei este funcţia ASC.

**Exemplu:**

```
PRINT CHR$(66)
B
```

Următorul exemplu este un program care tipăreşte pe ecran toate caracterele reprezentabile împreună cu codul lor ASCII:

```
10 CLS
20 FOR I = 1 TO 255
30 ' ignora toate caracterele nereprezentabile
40 IF (I > 6 AND I < 14) OR (I > 27 AND I < 32)
THEN 100
50 COLOR 0,7                    'negru pe alb
60 PRINT USING "###";I;         '3 cifre cod ASCII
70 COLOR 7,0                    'alb pe negru
80 PRINT " ";CHR$(I);" ";
90 IF POS(0) > 75 THEN PRINT
100 NEXT I
```

## CINT

Funcţie

**Scop:** Converteşte argumentul **x** într-un întreg.

**Format:** *v = CINT(x)*

**Comentariu:** *x* poate fi orice expresie numerică. Dacă *x* este în afara domeniului de la -32768 la 32767 va fi semnalată eroare de tip "**Overflow**". *x* este convertit la un întreg prin rotunjirea (în sus) a părţii fracţionare.

**Exemplu:** Observaţi, în exemplele urmatoare cum se realizează rotunjirea:

```
PRINT CINT(45.499)
45

PRINT CINT(-2.89)
-3
```

## CIRCLE

**Instrucţiune**

**Scop:** Desenează, pe ecran, o elipsă cu centrul (x,y) şi raza r.

**Format:** *CIRCLE(x,y),r[culoare[,început,sfîrşit[aspect]]]*

**Comentariu:** *(x,y)* reprezintă coordonatele elipsei. Acestea pot fi date atît sub forma absolută cît şi relativă.

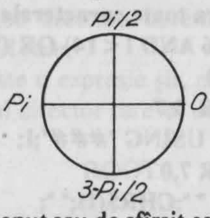
*r* este raza (axei majore), în puncte, a elipsei.

*culoare* este o expresie numerică întreagă care indică atributul de culoare din gama atributelor de culoare a modului ecran curent.

*început,sfîrşit* reprezintă unghiurile în radiani în domeniul - 2*PI la 2*PI, unde PI = 3.141593.

*aspect* este o expresie numerică.

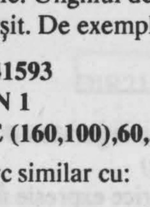
*început* şi *sfîrşit* specifică de unde va începe şi unde se va termina desenarea elipsei. Unghiurile sînt poziţionate în standardul matematic, cu 0 la dreapta şi crescător în sens trigonometric.




Dacă unghiul de început sau de sfîrşit este negativ, elipsa este legată de punctul de centru cu o linie. Unghiul de început poate fi mai mare sau mai mic decît unghiul de sfîrşit. De exemplu:

```
10 PI=3.141593
20 SCREEN 1
30 CIRCLE (160,100),60,,-PI,-PI/2
```

desenează o parte a unui cerc similar cu:



Argumentul *aspect* afectează raportul dintre raza mare şi raza mică. Dacă nu este specificată valoarea, aspectul este 5/6 în rezoluţie medie şi 5/12 în înaltă rezoluţie, din aceste valori rezultînd un raport de 4/3 pentru ecranul standard.

## CLEAR

**Comandă**

**Scop:** Atribuie tuturor variabilelor valoarea zero şi anulează toate variabilele şir. Opţiunile comenzii atribuie numărul maxim de octeţi pe care îi va adresa limbajul, precum şi mărimea stivei.

**Format:** *CLEAR[,[n][,m]]*

**Comentariu:** *n* specifică numărul maxim de octeţi ai segmentului de date BASIC. Dacă nu se specifică, se ia pentru *n* 65535. Se poate specifica o valoare mai mică pentru *n* pentru a descreşte spaţiul adresabil total BASIC. Aceasta face să crească, în schimb, totalul memoriei disponibile dincolo de segmentul de date BASIC în memoria superioară, operaţie necesară, de exemplu, pentru rezervarea de spaţiu de memorie pentru programele în limbaj masină.

*m* setează spaţiul stivei BASIC. Dacă nu se specifică, se ia valoarea de 512 octeţi, sau 1/8 din memoria disponibilă. Parametrul *m* trebuie inclus în cazul în care sînt folosite mai multe înlănţuiri de instrucţiuni GOSUB sau bucle FOR ... NEXT, sau în cazul în care se utilizează instrucţiunea PAINT pentru crearea unor scene complexe.

**CLEAR** eliberează întreaga memorie de date, fără a şterge însă programul curent.

**Exemple:**

**CLEAR ,32768**

şterge datele şi setează segmentul de date la 32 Kocteţi.

**CLEAR ,,2000**

şterge datele şi setează mărimea stivei la 2000 de octeţi.

## CLOSE

**Instrucţiune**

**Scop:** Încheie activitatatea de I/O (intrare-ieşire) cu un periferic sau fişier.

**Format:** *CLOSE [[#]nrfis[,[#]nrfis]...]*

**Comentariu:** *nrfis* este un număr utilizat în instrucţiunea OPEN. Asocierea între un fişier sau periferic particular şi un număr de fişier se încheie odată cu executarea instrucţiunii CLOSE. Dacă instrucţiunea CLOSE nu este urmată de nici un parametru, vor fi închise toate fişierele sau perifericele.

**Exemplu:**

**9900 CLOSE #1,#2,#3**

## CLS

Instrucţiune

**Scop:** Şterge ecranul.

**Format:** *CLS*

**Comentariu:** Dacă ecranul este în modul text, este ştearsă pagina activă. Dacă ecranul este în modul grafic este ştearsă toată memoria tampon de ecran. Instrucţiunea CLS aduce, de asemenea cursorul în poziţia iniţiala, stînga-sus.

## COLOR

Instrucţiune

**Scop:** Atribuie culorile pentru fundal ,prim-plan şi margine. Sintaxa instrucţiunii COLOR depinde de modul de lucru (text sau grafic), mod care a fost setat cu ajutorul instrucţiunii SCREEN.

**In modul text:**

| | |
|---|---|
| **prim-plan** | 1 la 16 atribute de culoare sau caracter clipitor. |
| **fundal** | 1 la 8 atribute de culoare. |
| **margine** | 1 la 16 atribute de culoare. |

**In modul grafică medie rezoluţie:**

| | |
|---|---|
| **fundal** | 1 la 16 atribute de culoare |
| **paleta** | 1 la 2 palete cu trei culori fiecare. |
| **margine** | la fel ca la fundal |

## COLOR (modul Text)

**Format:** *COLOR [primplan][,[fundal][,margine]*

**Comentariu:** *primplan* este o expresie numerică cu valori între 0 şi 31, reprezentînd culoarea caracterului.

*fundal* este o expresie numerică cu valori între 0 şi 7, pentru fundal.

*margine* este o expresie numerică cu valori între 0 şi 15, pentru marginea ecranului.

Culorile permise pentru prim-plan sînt:

| | |
|---|---|
| 0 negru | 8 gri |
| 1 albastru | 9 albastru deschis |
| 2 verde | 10 verde deschis |
| 3 bleu | 11 bleu deschis |
| 4 roşu | 12 roşu deschis |
| 5 magenta | 13 magenta deschis |
| 6 maro | 14 galben |
| 7 alb | 15 alb strălucitor |

Caracterele pot fi făcute să clipească prin adăugarea de 16 unităţi la

valoarea culorii alese.

Pentru fundal pot fi alese culorile de la 0 la 7.

Cu adaptorul de grafică monocrom paralel, pentru caractere pot fi folosite următoarele culori:

0 negru

1 caracter subliniat

2-7 culoarea standard primplan

Pentru adaptorul grafic/color, adăugînd 8 la numărul culorii dorite, se va obţine aceeaşi culoare în intensitate mărită. Dacă la culoarea dorită se adaugă 16 se obţine aceeaşi culoare, dar clipitoare.

**În modul grafic:**

**Format:** *COLOR [fundal][,[paleta]]*

**Comentariu:** *fundal* este o expresie întreagă cu valori între 0 şi 15. Specifică atributul de culoare al fundalului.

*paleta* este o expresie întreagă. Selectează una din cele doua palete de culoare.

În modul grafic, instrucţiunea COLOR atribuie culoarea fundalului şi alege una din cele două palete, fiecare cu cîte 4 atribute de culoare (0-3). Atributul de culoare 0 este totdeauna culoarea fundalului curent. Instrucţiunea COLOR are înţeles, în modul grafic, doar pentru rezoluţia medie. Utilizarea acestei instrucţiuni în rezoluţie înaltă duce la semnalarea unei erori de tip "**Illegal function call**". Culorile care pot fi selectate sînt următoarele:

| **Culoare** | **Paleta 1** | **Paleta 0** |
|---|---|---|
| 1 | verde | bleu deschis |
| 2 | roşu | magenta |
| 3 | maro | alb |

Dacă paleta este un număr par, va fi selectată paleta 0, iar dacă paleta este un număr impar oarecare, se va selecta paleta 1.

Schimbarea culorii prim-plan a caracterelor de la 3 la 2 la 1 se poate realiza introducînd:

**DEF SEG: POKE &HFE,CULOARE**

unde CULOARE este atributul de culoare dorit (1, 2 sau 3).

## COM(n)

Instrucţiune

**Scop:** Activează sau dezactivează comunicaţiile prin adaptorul de comunicaţii specificat.

**Format:** *COM(n) ON*

*COM(n) OFF*
*COM(n) STOP*

**Comentariu:** *n* este numărul adaptorului de comunicaţie (1 sau 2).

Instrucţiunea **COM(n) ON** trebuie executată înainte de lansarea instrucţiunii ON COM(n). Dacă **COM(n)** este **OFF** nu se poate realiza activitatea de comunicaţii, aceasta fiind declarată închisă. Instrucţiunea **COM(n) STOP** dezafectează, pentru moment, linia de transmisie, dar fără închiderea definitivă a activităţii, urmînd ca la lansarea instrucţiunii **COM(n) ON** aceeaşi activitate să fie reluată.

## COMMON

Instrucţiune

**Scop:** Trece variabilele către un program de legătură.

**Format:** *COMMON var[,var]...*

**Comentariu:** *var* este numele variabilei ce urmează a fi trecută în programul de legătură. Vectorii sînt specificaţi prin adăugarea "( )" la numele vectorului.

Instrucţiunea **COMMON** poate apare oriunde în program, dar este preferabil să fie amplasată la începutul acestuia. Se pot folosi oricîte instrucţiuni COMMON, dar este obligatoriu ca variabilele să fie numite o singură dată. Dacă în programul de legătură sînt necesare toate variabilele din programul curent, instrucţiunea CHAIN se va folosi cu opţiunea ALL şi va fi omisă utilizarea instrucţiunii **COMMON**.

**Exemplu:**

```
100 COMMON A,B1,C,D(),S$
110 CHAIN "A:PROG3"
```

## CONT

Comandă

**Scop:** Reia execuţia unui program după o comandă de "break".

**Format:** *CONT*

**Comentariu:**

Comanda **CONT** poate fi utilizată pentru reluarea execuţiei unui program, după ce programul a fost oprit cu Ctrl-Break. Execuţia continuă de la punctul în care a avut loc întreruperea.

Comanda **CONT** este de obicei utilizată cu comanda STOP pentru depanarea programelor. Cînd execuţia este oprită, valorile variabilelor pot fi examinate sau schimabate utilizînd instrucţiunile de mod direct.

Comanda **CONT** nu este executabilă dacă programul sau linii din programul curent au fost editate.

## COS

Funcţie

**Scop:** Calculează funcţia cosinus trigonometric.

**Format:** *v = COS(x)*

**Comentariu:** *x* este unghiul al cărui cosinus urmează a fi calculat. Valoarea lui *x* trebuie să fie în radiani.

**Exemplu:**

```
10 PI=3.141593
20 GRAD=180
30 RAD=GRAD*PI/180
40 PRINT COS(RAD)
RUN
-1
```

## CSNG

Funcţie

**Scop:** converteşte argumentul **x** în simplă precizie.

**Format:** *v = CSNG(x)*

**Comentariu:** *x* este o expresie numerică ce va fi convertită în simplă precizie.

**Exemplu:**

```
10 A#=975.3421222#
20 PRINT A#; CSNG(A#)
RUN
975.3421222 975.3421
```

## CSRLIN

Variabilă

**Scop:** Citeşte coordonata verticală a cursorului.

**Format:** *v = CSRLIN*

**Comentariu:** Variabila **CSRLIN** redă poziţia cursorului liniei (rîndului) curente în pagina activă, putînd lua valori între 1 şi 25. Funcţia POS redă poziţia cursorului coloanei. Vezi "Funcţia POS".

**Exemplu:** În acest exemplu se salvează coordonatele cursorului în variabilele X şi Y, apoi cursorul este mutat pe linia 24, unde este afişat cuvîntul "SCOR =", după care cursorul este mutat la vechea poziţie.

```
10 Y=CSRLIN 'salveaza linia curenta
20 X=POS(0) 'salveaza coloana curenta
30 LOCATE 24,1: PRINT "SCOR ="
40 LOCATE Y,X    'inapoi la vechea pozitie
```

## CVI,CVS,CVD

Funcţii

**Scop:** Convertesc variabilele şir în variabile numerice.

**Format:** *v = CVI(şir 2 octeţi)*
*v = CVS(şir 4 octeţi)*
*v = CVD(şir 8 octeţi)*

**Comentariu:** Valorile numerice citite dintr-un fişier aleator trebuie convertite din şiruri în numere. CVI converteşte un şir de 2 octeţi într-un întreg. CVS converteşte un şir de 4 octeţi într-un număr simplă precizie. CVD converteşte un şir de 8 octeţi într-un număr dublă precizie.

**Exemplu:** Acest exemplu utilizează un fişier aleator (#1) care are cîmpurile definite în linia 100. Linia 110 citeşte o înregistrare din fişier. Linia 120 utilizează funcţia CVS pentru interpretarea primilor 4 octeţi (N$) ai înregistrării, ca un număr în simplă precizie. N$ era la origine, probabil, un număr scris în fişier utilizînd funcţia MKS$.

```
100 FIELD #1,4 AS N$, 12 AS B$
110 GET #1
120 Y=CVS(N$)
```

## DATA

Instrucţiune

**Scop:** Memorează constantele şir şi numerice ce sînt adresate de instrucţiunea READ.

**Format:** *DATA const[,const]...*

**Comentariu:** *const* poate fi orice constantă şir sau numerică.

În lista constantelor nu sînt permise expresii. Constantele numerice pot fi de orice formă - întregi, virgulă fixă, virgulă mobilă, hexazecimale sau octale. Constantele şir nu trebuie incluse între ghilimele atîta timp cît nu conţin virgule, două puncte sau spaţii semnificative.

Instrucţiunile DATA nu sînt executabile şi pot fi plasate oriunde în program. O instrucţiune DATA poate conţine oricîte constante în limita lungimii unei linii de program, iar un program poate conţine oricîte instrucţiuni DATA. Instrucţiunea READ accesează instrucţiunile DATA în ordinea numărului de linie. Pentru recitirea informaţiilor dintr-o instrucţiune DATA trebuie folosită instrucţiunea RESTORE.

**Exemplu:** Vezi exemplul de la instrucţiunea READ.

## DATE$

Variabilă şi Instrucţiune

**Scop:** Instaurează sau alocă data calendaristică.

**Format:**

Ca variabilă: *v$ = DATE$*

Ca instrucţiune: *DATE$ = x$*

**Comentariu:** Pentru variabilă (v$ = DATE$):

Este alocat un şir de 10 caractere în forma **ll-zz-aaaa** unde:

*ll* este luna;

*zz* este ziua;

*aaaa* este anul.

Pentru instrucţiune (**DATE$ = x$**):

*x$* este o expresie şir utilizată pentru instaurarea datei calendaristice curente. *x$* poate lua una din formele:

ll-zz-aa

ll/zz/aa

ll-zz-aaaa

ll/zz/aaaa

Anul poate lua valori în domeniul 1980-2099.

**Exemplu:**

```
10 DATE$ = "12/31/1988"
20 PRINT DATE$
RUN
12-31-1988
```

## DEF FN

Instrucţiune

**Scop:** Defineşte şi numeşte o funcţie.

**Format:** *DEF FNnume[(arg,[arg]...)] = expresie*

**Comentariu:** nume este numele variabilei. Aceste nume precedate de prefixul **FN** devin numele funcţiei.

*arg* este un argument. Reprezintă numele unei variabile din definiţia funcţiei, ce va fi înlocuit cu o valoare în momentul apelării funcţiei. Argumentele din listă reprezintă, într-o bază unu-la-unu, valorile ce vor fi atribuite în momentul apelării funcţiei.

*expresie* defineşte şi redă valoarea funcţiei. Tipul expresiei (numerică sau şir) trebuie să coincidă cu tipul declarat în *nume*.

Definirea unei funcţii este limitată la o singură instrucţiune. Argumentele (*arg*) ce apar în definirea funcţiei servesc doar la definiţie; ele nu afectează variabilele de program ce au acelaşi nume. Un nume de variabilă utilizat în *expresie* nu trebuie să apară în lista argumentelor, altfel, chiar această valoare va fi furnizată în momentul apelării.

Instrucţiunea **DEF FN** trebuie executată înaintea apelării funcţiei pe

care o defineşte.

Nu sînt suportate funcţiile recursive.

DEF FN nu este permisă în modul direct.

**Exemplu:**

```
10 PI=3.141593
20 DEF FNARIA(R)=PI*R^2
30 INPUT "Raza? ",RAZA
40 PRINT "Aria este ";FNARIA(RAZA)
RUN
Raza?
```

(Să spunem că răspundem cu 2)

```
Raza?2
Aria este 12.56637
```

Şi acum un exemplu de două argumente:

```
10 DEF FNMUD(X,Y)=X-(INT(X/Y)*Y)
20 A = FNMUD(7.4,4)
30 PRINT A
3.4
```

## DEF SEG

Instrucţiune

**Scop:** Defineşte segmentul curent de memorie.

**Format:** *DEF SEG[ =segment]*

**Comentariu:** *segment* este o expresie numerică cu valori între 0 şi 65535.

Valoarea iniţială a segmentului, la începutul sesiunii BASIC, este valoarea segmentului de date (DS) BASIC, sau, cu alte cuvinte, începutul spaţiului de memorie de lucru a utilizatorului. Valoarea segmentului de date BASIC poate fi găsită la segmentul 0, offset &H510 şi &H11. Dacă se dă valoarea segmentului, aceasta trebuie să fie cuprinsă în limite de cîte 16 octeţi, deoarece segmentele încep doar în limitele paragrafelor. Valoarea este deplasată la stînga 4 biţi (multiplicat cu 16) pentru a forma adresa segmentului operaţiei următoare. Aceasta înseamnă că, dacă segmentul este în hexazecimal, este adăugat un 0 (zero) pentru a da adresa actuală a segmentului. Limbajul BASIC nu execută nici o verificare pentru validarea valorii segmentului.

Cuvintele DEF si SEG trebuie separate printr-un spaţiu; altfel instrucţiunea DEFSEG=100 va fi interpretată "atribuie valoarea 100 variabilei DEFSEG".

**Exemplu:** Primul exemplu restaurează un segment în segmentul de

date BASIC:

**DEF SEG**

În al doilea exemplu, segmentul este atribuit memoriei tampon a adaptorului de grafică color (&HB800, offset 0).

**DEF SEG = &HB800**

## DEFtip

Instrucţiune

**Scop:** Declară tipul varibilei ca întreg, simplă precizie, dublă precizie sau şir.

**Format:** *DEFtip literă[-literă][,literă[-literă]]...*

**Comentariu:** *tip* este INT, SNG, DBL sau STR.

*literă* este o literă a alfabetului (A-Z).

O instrucţiune DEFtip declară că variabilele începînd cu litera sau literele specificate vor fi de tipul descris. Totuşi, caracterele de declarare a tipului (%, !, #, $) au, întotdeauna, precedenta mai mare asupra instrucţiunii DEFtip în declararea unei variabile. Dacă nu se specifică tipul unei variabile, se presupune a fi în simplă precizie.

**Exemplu:**

```
10 DEFDBL L-P
20 DEFSTR A
30 DEFINT D-H,X
40 NUMAR = 1#/3: PRINT NUMAR
50 ANIMAL = "MOTAN": PRINT ANIMAL
60 X = 10/3: PRINT X
RUN
.3333333333333333
MOTAN
3
```

## DEF USR

Instrucţiune

**Scop:** Specifică locaţia din memorie a unei subrutine în limbaj masină, care va fi, ulterior, apelată de funcţia USR.

**Format:** *DEF USR[n] = offset*

**Comentariu:** *n* trebuie să fie un număr între 1 şi 9. Identifică numărul subrutinei USR. Dacă se omite *n*, se presupune DEF USR0.

*offset* este o expresie întreagă cuprinsă între 0 şi 65535. Valoarea offset-ului este adăugată la valoarea segmentului curent pentru a obţine adresa curentă de start a rutinei USR.

**Exemplu:** Acest exemplu încarcă o subrutină limbaj masină într-un

vector întreg. Valoarea lui n din linia 60 este determinată de mărimea subrutinei.

```
10 OPTION BASE 1
20 DEFINT A-Z
30 'Se definesc toate variabilele de VARPTR
40 SUBRT = 0: I=0: J=0
50 'Dimensionarea vectorului pentru subrutina
60 DIM VECTOR(N)
70 'Obtinem offset-ul primului element al vectorului
in segmentul de date BASIC
80 SUBTR = VARPTR(VECTOR(1))
90 'Incarca rutina in vector
100 BLOAD "ASMFILE",SUBTR
.
.
.
1000 'Trece offset-ul în DEF USR
1010 DEFUSR0 = VARPTR(VECTOR(1))
1020 'Executa subrutina
1030 J = USR0(I)
```

## DELETE

Comandă

**Scop:** Şterge linii de program.

**Format:** *DELETE [linie1][-linie2]*
*DELETE [linie1-]*

**Comentariu:** *linie1* este numărul primei linii ce urmează a fi ştearsă. *linie2* este numărul ultimei linii ce urmează a fi ştearsă.

**DELETE line1-**

şterge toate liniile de la linia specificată la sfîrşitul programului.

**DELETE -linie1**

şterge toate liniile de la începutul programului la linia specificată.

Poate fi utilizat un punct (.) în locul unui număr de linie pentru a indica linia curentă de program.

**Exemple:**

**DELETE 10**

şterge linia 10 din program;

**DELETE 20-70**

şterge toate liniile cuprinse între linia 20 şi linia 70;

**DELETE .**

Presupunînd că ultima linie editată a fost 220, comanda de mai sus şterge această linie;

**DELETE 100-**

şterge toate liniile de program începînd cu linia 100, inclusiv, pînă la sfîrşitul programului;

**DELETE -80**

şterge toate liniile de program, de la început pînă la linia 80 inclusiv.

## DIM

Instrucţiune

**Scop:** Specifică valoarea maximă a elementelor unei matrici (vector) şi alocă memorie în concordanţă cu numărul de elemente.

**Format:** *DIM variabilă(elemente)[,varabilă(elemente)]...*

**Comentariu:** *variabilă* este numele matricii.

*element* este o listă de expresii numerice, separate prin virgule, elemente care definesc dimensiunile matricii.

La execuţie, instrucţiunea DIM atribuie, iniţial, valoarea zero tuturor elementelor matricii. Elementele unei matrici şir sînt, toate, de lungime variabilă, cu valoare iniţială nulă (şir de lungime zero).

Dacă numele unei variabile vectoriale este utilizat fără a se fi declarat dimensiunea acesteia cu ajutorul instrucţiunii DIM, numărul maxim de elemente este presupus a fi de 10. Numărul minim de elemente al unei matrici este totdeauna 0, în afara cazului cînd a fost menţionat expres cu ajutorul instrucţiunii OPTION BASE. Numărul maxim de dimensiuni pentru o matrice este 255.

Redimensionarea unei matrici este prohibită, ea putîndu-se realiza, în cazul unui program doar după execuţia instrucţiunii ERASE.

**Exemplu:** Acest exemplu crează două matrici; una unidimensională (SIS) cu 13 elemente şi una bidimensională (LIT$)

```
10 MAX=2
20 DIM SIS(12), LIT(MAX,2)
30 DATA 26.5, 37, 8, 29, 80, 9.9, &H800
40 DATA 7, 18, 55, 12, 5, 43
50 FOR I=0 TO 12
60 READ SIS(I)
70 NEXT I
80 DATA ION, VASILE, "A:"
90 DATA "Buna ziua", SALUT, Bonjour
100 DATA NOAPTE BUNA, BUCURESTI, PLOIESTI
```

```
110 FOR I=0 TO 2: FOR J=0 TO 2
120 READ LIT$(I,J)
130 NEXT J,I
140 PRINT SIS(3); LIT$(2,0)
RUN
29 NOAPTE BUNA
```

## DRAW

Instrucţiune

**Scop:** Desenează un obiect specificat într-un şir de caractere.

**Format:** *DRAW şir*

**Comentariu:** Instrucţiunea DRAW desenează obiecte utilizînd un limbaj de definiţii grafice. Comenzile limbajului sînt conţinute într-o expresie şir. În timpul execuţiei BASIC-ul examinează valorile şirului. Cînd este dată o comandă de deplasare, se desenează o linie de la ultimul punct de referinţă. Valoarea lui *n* din următoarele comenzi de deplasare indică mărimea deplasării. Numărul de puncte de deplasare este *n* ori factorul de scalare. (atribuit prin comanda **S**). În continuare, sînt prezentate, detaliat, comenzile de deplasare.

**Un** Deplasare în sus.

**Dn** Deplasare în jos.

**Ln** Deplasare la stînga.

**Rn** Deplasare la dreapta.

**En** Deplasare pe diagonală sus şi dreapta.

**Fn** Deplasare pe diagonală jos şi dreapta.

**Gn** Deplasare pe diagonală jos şi stînga.

**Hn** Deplasare pe diagonală sus şi stînga.

**Mx,y** Deplasare absolută sau relativă. Dacă x are un semn plus ( + ) sau un semn minus (-) în faţa sa, mişcarea este relativă, în alt caz este absolută.

Următoarele două prefixe de comandă pot precede oricare din aceste comenzi:

**B** Deplasare, dar fără plotare.

**N** Deplasare, dar cu întoarcere în poziţia iniţială.

Mai sînt disponibile următoarele comenzi:

**An** Atribuie unghiul *n*. Valoarea lui *n* poate varia între 0 şi 3, unde 0 înseamnă 0 grade, 1 înseamnă 90 grade, 2 înseamnă 180 grade, iar 3, 270 grade. Figurile rotate cu 90 şi, respectiv, 270 grade sînt scalate, astfel încît

ele apar ca fiind de aceeaşi dimensiune cu cele la 0 şi 180 grade, pe un display standard care are raportul de aspect 4/3.

**TAn** Întoarcere cu unghiul *n*. Valoarea lui *n* este cuprinsă între - 360 şi 360. Dacă *n* este pozitiv ( + ), unghiul de întoarcere este contrar sensului acelor de ceasornic şi invers pentru *n* negativ (-).

**Cn** Atribuie culoarea *n*. Valoarea lui *n* este cuprinsă între 0 şi 3 în medie rezoluţie, şi 0 sau 1 în rezoluţie înaltă. În medie rezoluţie *n* selectează atributul de culoare specificat de instrucţiunea COLOR. Zero (0) este totdeauna atributul pentru fundal. Atributul iniţial de culoare pentru prim-plan este atributul maxim de culoare pentru modul ecran curent: 3 în medie rezoluţie şi 1 în rezoluţie înaltă.

**Sn** Atribuie factorul de scală. Valoarea lui n poate varia între 1 şi 255. Factorul de scală este *n* împărţit la 4. De exemplu, dacă n = 1, atunci factorul de scală este 1/4. Valoarea iniţiala este 4, deci factorul de scală este 1.

**X variabilă;** Execută subşirul. Aceasta permite execuţia unui al doilea şir în cadrul primului.

**P m,n** Setează culoarea figurii la *m* şi culoarea ramei la *n*.

În toate comenzile descrise mai sus, argumentele m,n,x sau y pot fi constante (de ex. 123) sau variabile **= variabilă;** unde *variabilă* este numele unei variabile numerice. Pentru utilizarea unei astfel de variabile sau în cazul comenzii X, este necesar ca, după numele variabilei, să apară semnul (;). Se pot, deasemenea, specifica nume de variabile sub forma **VARPTR$(variabilă)**, în loc de **= variabilă;**. De exemplu:

**DRAW "XA$;"** se poate scrie **DRAW "X" + VARPTR$(A$)**

**DRAW "S = SC;"** se poate scrie **DRAW "S = "; + VARPTR$(SC)**

Raportul de aspect al ecranului determină spaţiul între punctele orizontale, verticale şi diagonale. Instrucţiunea DRAW nu ţine cont de raportul de aspect al modului ecran curent; astfel, **DRAW "R50 U50"**, va plota 50 de puncte la dreapta şi 50 puncte în sus, dar liniile rezultate nu au lungime egală. Explicaţia rezidă din faptul că ecranul nu are un număr egal de puncte pe orizontală şi pe verticală. Astfel, pentru a calcula raportul de aspect vom împărţi numărul de puncte pe orizontală cu numărul de puncte pe verticală. Deci:

- pentru medie rezoluţie : 320 puncte/200 puncte = **5/6**
- pentru înaltă rezoluţie: 640 puncte/320 puncte = **5/12**

Dacă dorim să desenăm un pătrat în oricare din modurile de rezoluţie trebuie scalată axa y cu raportul de aspect corespunzător, sau axa x 1/raportul de aspect.

Pentru a desena un pătrat cu latura de 100 se scalează axa x cu:

**10 '100*6/5 inseamna 120**

**20 DRAW "U100 R120 D100 L120"**

## EDIT

Comandă

**Scop:** Afişează o linie pentru editare.

**Format:** *EDIT linie*

**Comentariu:** *linie* este numărul unei linii existente în program.

Comanda **EDIT** afişează linia pe ecran poziţionînd cursorul pe primul caracter al liniei.

## END

Instrucţiune

**Scop:** Termină execuţia programului, închide toate fişierele şi dă controlul nivelului de comandă.

**Format:** *END*

**Comentariu:** Instrucţiunea **END** poate fi plasată oriunde într-un program pentru a termina execuţia. **END** diferă de STOP în două moduri:

- END nu cauzează tipărirea mesajului **Break.**
- END închide toate fişierele.

Instrucţiunea END la sfîrşitul programului este opţională.

**Exemplu:** În exemplu de mai jos, programul se termină dacă variabila K este mai mare decît 100; altfel, programul face salt la linia 20.

**100 IF K > 100 THEN END ELSE GOTO 20**

## ENVIRON

Instrucţiune

**Scop:** Modifică parametrii tabloului de referinţe globale BASIC. **ENVIRON** este utilizat pentru a schimba parametrul "PATH" a unui proces descendent sau de a trece parametrii unui proces descendent prin inventarea unui nou tabel de referinţe globale.

**Format:** *ENVIRON param = şir*

**Comentariu:** *param* este numele unui parametru ca de exemplu "PATH".

*şir* este textul care defineşte noul parametru.

*param* trebuie separat de *şir* prin semnul egal sau printr-un spaţiu. ENVIRON ia drept *param* toate caracterele pînă la primul spaţiu. Primul caracter care nu este spaţiu sau semnul egal după *param* este luat drept *şir*.

**Exemplu:** Pentru a crea o cale de acces spre directorul rădăcină din

cititorul de disc A, folosim următoarea instrucţiune:

**ENVIRON "PATH=A:\"**

## ENVIRON$

Funcţie

**Scop:** Recuperează şi afişează şirul specificat din tabelul de referinţe globale BASIC.

**Format:** *v$=ENVIRON$(param)*

**sau** *v$=ENVIRON$(n)*

**Comentariu:** *param* este o expresie şir conţinînd parametrii de recuperat.

*n* este o expresie întreagă cu valori între 1 şi 255.

Dacă este utilizat un argument şir, **ENVIRON$** recuperează, din tabelul de referinţe globale, un şir conţinînd textul dat de *param*.

În cazul în care este utilizat un argument numeric, ENVIRON$ recuperează un şir conţinînd al *n*-lea parametru din tabelul de referinţe globale, pentru tot şirul *param* = text.

**ENVIRON$** face distincţie între majuscule şi minuscule.

**Exemplu:** În momentul încărcării iniţiale, sistemul de operare DOS atribuie un parametru numit "COMSPEC" care indică sistemului de operare unde să localizeze fişierul COMMAND.COM şi atribuie o cale de acces nulă. Pentru a observa conţinutul tabelului de referinţe globale la momentul iniţializării, se introduce următoarea instrucţiune BASIC:

**PRINT ENVIRON$(1)**

ceea ce va avea drep rezultat tipărirea pe ecran:

**PATH=**

Dacă se introduce:

**PRINT ENVIRON$(2)**

vom vedea afişat:

**COMSPEC=A:\COMMAND.COM**

Dar dacă se introduce:

**PRINT ENVIRON$("COMSPEC")**

răspunsul calculatorului va fi:

**A:COMMAND.COM**

Programul următor salvează tabelul de referinţe globale BASIC într-un vector, acest tabel putînd fi modificat într-un proces descendent. După ce acest proces este încheiat, referinţele globale sînt realocate:

```
10 DIM TABEL$(10) 'atribuie 10 parametri
20 PARAM=1 'numarul initial de parametri
30 WHILE LEN(ENVIRON$(PARAM))>0
```

```
40 TABEL$(PARAM) = ENVIRON$(PARAM)
50 PARAM = PARAM + 1
60 WEND
70 PARAM = PARAM-1 'ajusteaza parametrii
80 'acum se salveaza noile referinte globale
90 ENVIRON "DATAIN = C:\DATAIN\INP.FIL"
100 ENVIRON
"SORT.DAT = SORT.DAT < " + ENVIRON$("DATAIN"
) + " > LPT1:"
.
.
.
1000 SHELL ENVIRON$("SORT.DAT")
1010 FOR I = 1 TO PARAM
1020 ENVIRON TABEL$(I) 'resalveaza parametrii
1030 NEXT I
.
.
.
```

## EOF

Funcţie

**Scop:** Indică condiţia de sfîrşit de fişier (**end-of-file**).

**Format:** *v = EOF(numfis)*

**Comentariu:** *numfis* este numărul fişierului specificat în instrucţiunea OPEN.

Funcţia **EOF** este utilă în evitarea erorii de tip **Input past end**. **EOF** întoarce valoarea -1 (adevărat) dacă a fost întîlnit sfîrşitul fişierului şi 0 (zero) dacă sfîrşitul nu a fost identificat.

**EOF** are semnificaţie doar pentru fişierele deschise pentru intrare de pe disc sau pentru un fişier de comunicaţie. O valoare de -1 pentru fişierele de comunicaţie înseamnă că buffer-ul este gol.

**Exemplu:** Acest exemplu citeşte informaţii de la un fişier secvenţial "DATE". Valorile sînt citite într-o matrice M pînă în momentul în care este depistat sfîrşitul fişierului:

```
10 OPEN "DATE" FOR INPUT AS #1
20 C = 0
30 IF EOF(1) THEN END
40 INPUT #1,M(C)
50 C = C + 1: GOTO 30
```

## ERASE

**Istrucţiune**

**Scop:** Elimină matrici din program.

**Format:** *ERASE numemat[,numemat]...*

**Comentriu:** *numemat* este numele matricii de şters.

După ştergerea matricii din memorie spaţiul alocat pentru acea matrice poate fi utilizată în alte scopuri.

Instrucţiunea **ERASE** este utilă pentru redimensionarea matricilor într-un program. Dacă se redimensionează matricile fără a se utiliza instrucţiunea **ERASE**, va fi semnalată eroare de tip **Duplicate definition**.

**Exemplu:**

```
10 START=FRE("")
20 DIM MARE(100,100)
30 MIJLOC=FRE("")
40 ERASE MARE
50 DIM MARE(10,10)
60 FINAL=FRE("")
70 PRINT START, MIJLOC, FINAL
62808          21980          62289
```

## ERDEV si ERDEV$

Variabile

**Scop:** Variabile de citire. Reţine codul de eroare generat de **INTerrupt 24** a unei erori de periferic, precum şi numele perifericului care a generat eroarea.

**Format:** *v=ERDEV*

*v$=ERDEV$*

**Comentariu:** În momentul detectării în sistemul de operare DOS a unei erori, **ERDEV** reţine codul de eroare generat de **INTerrupt 24** în cei mai de jos 8 biţi, iar cei mai de sus 8 biţi conţin biţii 13, 14 şi 15 pentru atributul blocului de început al perifericului. **ERDEV$** conţine în două caractere numele perifericului (A:, B:, etc).

**Exemplu:** Deschizînd fereastra cititorului de disc B: şi introducînd:

**FILES "B:"**

se va afişa mesajul:

**Disk not ready**

şi introducînd:

**PRINT ERDEV, ERDEV$**

se obţine:

**2          B:**

Exemplul următor simulează o eroare la imprimantă:

```
10 CLS
20 ON ERROR GOTO 60
30 LPRINT "Imprimanta este pregatita"
40 PRINT "Imprimanta este pregatita"
50 END
60 V$=HEX$(ERDEV)
70 PRINT "ERDEV = ";V$
80 D$=ERDEV$
90 PRINT "ERDEV$ = ";D$
100 RESUME NEXT
```

Dacă se rulează acest exemplu cu imprimanta oprită, calculatorul va afişa:

```
ERDEV = 8009
ERDEV$ = LPT1
```

## ERR si ERL

Variabile

**Scop:** Atribuie unei variabile codul de eroare şi numărul liniei asociate cu o eroare.

**Format:** *v=ERR*
*v=ERL*

**Comentariu:** Variabila **ERR** conţine codul de eroare pentru ultima eroare, iar variabila **ERL** conţine numărul liniei de program în care a fost semnalată eroarea.

**Exemplu:** Acest exemplu testează dacă fereastra cititorului de disc este deschisă:

```
10 ON ERROR GOTO 100
20 OPEN "DATE" FOR INPUT AS #1
30 END
.
.
.
100 IF ERR=71 THEN LOCATE 23,1
110 PRINT "FEREASTRA DESCHISA !": RESUME
```

## ERROR

Instrucţiune

**Scop:** Simulează apariţia unei erori, sau permite definirea unor coduri de erori ale utilizatorului.

**Format:** *ERROR n*

**Comentariu:** *n* trebuie să fie cuprins între 0 şi 255.

Dacă *n* are aceeaşi valoare cu un cod de eroare folosit de limbajul BASIC, instrucţiunea ERROR simulează apariţia acelui tip de eroare. Dacă a fost definită o rutină de manipulare a erorilor de către instrucţiunea ON ERROR, este introdusă acea rutină de eroare. Altfel, este afişat mesajul de eroare corespunzător codului şi se opreşte execuţia programului. Pentru definirea unor erori ale utilizatorului este necesară utilizarea unor valori diferite de cele BASIC.

**Exemplu:** Primul exemplu simulează o eroare de tip **String too long**:

```
10 T=15
20 ERROR T
RUN
String too long în line 20
```

Exemplul următor generează un cod de eroare (210) care nu este utilizat de BASIC:

```
100 ON ERROR GOTO 1000
110 INPUT "Deplasarea pe axa X = ",X
120 IF X>639 THEN ERROR 210
.
.
.
1000 IF ERR=210 THEN PRINT
"Limita pentru axa X = 639"
1010 IF ERL 120 THEN RESUME 110
```

## EXP

Funcţie

**Scop:** Calculează funcţia exponenţială.

**Format:** *v=EXp(x)*

**Comentariu:** *x* poate fi orice expresie numerică.

Funcţia atribuie variabilei *v* numărul matematic **e** ridicat la puterea *x*, unde **e** este baza logaritmilor naturali. Dacă *x* este mai mare decît 88.02969 se va semnala eroare.

**Exemplu:**

```
10 X=2
20 PRINT EXP(X-1)
RUN
2.718282
```

## FIELD

Instrucţiune

**Scop:** Alocă spaţiu variabilelor dintr-un fişier aleator.

**Format:** *FIELD [#nrfis,lung AS varsir[,lung AS varsir]...*

**Comentariu:** *nrfis* este numărul sub care a fost deschis fişierul.

*lung* este o expresie numerică ce specifică numărul de caractere alocat variabilei *varsir*.

*varsir* este o variabilă şir utilizată pentru accesarea fişierului aleator.

**Exemplu:**

```
10 OPEN "FOO" AS #1
20 FIELD 1, 100 AS A#, 200 AS B$
30 FIELD 1, 300 AS DMY$, 40 AS C$
```

## FILES

Comandă

**Scop:** Afişează numele fişierelor din directorul discului curent.

**Format:** *FILES [filespec]*

**Comentariu:** *filespec* este o expresie şir. Dacă se omite *filespec* sînt afişate toate fişierele de pe directorul curent. Regulile de citire sînt aceleaşi ca în cazul comenzii **dir** din sistemul de operare DOS.

## FIX

Funcţie

**Scop:** Trunchiază *x* la un întreg.

**Format:** *v=FIX(x)*

**Comentariu:** *x* poate fi orice expresie numerică. Diferenţa între FIX şi INT este că, în cazul funcţiei **FIX**, nu este evaluat următorul număr mai mic pentru un *x* negativ.

**Exemple:**

```
PRINT FIX(33.82)
33

PRINT FIX(-2.77)
-2
```

## FOR şi NEXT

Instrucţiuni

**Scop:** Execută o serie de instrucţiuni într-o buclă, de un număr dat de ori.

**Format:** *FOR var=x TO y [STEP z]*

.

.

.

**NEXT [var[,var]...]**

**Comentariu:** *var* este o variabilă întreagă sau simplă precizie utilizată drept contor.

*x* este o expresie numerică care reprezintă valoarea inițială a contorului.

*y* este o expresie numerică care reprezintă valoarea finală a contorului.

*z* este o expresie numerică utilizată drept increment.

Liniile de program care urmează instrucțiunii **FOR** sînt executate pînă la întîlnirea instrucțiunii NEXT. Apoi contorul este incrementat cu valoarea incrementului *z*. Dacă nu se specifică o valoare pentru *z*, incrementarea se face cu o unitate. După incrementare se realizează o verificare pentru a se vedea dacă valoarea contorului nu este mai mare decît valoarea finală. Dacă nu este mai mare, BASIC-ul face salt la instrucțiunea imediat următoare instrucțiunii **FOR** și procesul se repetă. În cazul în care contorul este mai mare decît valoarea finală, se dă controlul instrucțiunii care urmează instrucțiunii NEXT.

Dacă *z* este negativ se execută decrementarea contorului pînă cînd acesta este mai mic decît valoarea finală.

Buclele **FOR-NEXT** pot fi întrepătrunse, aceasta însemnînd ca o buclă poate fi plasată în interiorul alteia. Dacă buclele sînt întrepătrunse, fiecare din ele trebuie să aibă un nume de variabială de contor unic. Instrucțiunea **NEXT** pentru o buclă interioară trebuie să apară înaintea instrucțiunii **NEXT** pentru o buclă exterioară. Dacă buclele întrepătrunse au același punct de terminare poate fi utilizată o singură instrucțiune **NEXT** pentru toate. De exemplu, o instrucțiune de forma:

**NEXT var1, var2, var3,...**

este echivalentă cu:

**NEXT var1**
**NEXT var2**
**NEXT var3**

.

.

.

Variabila sau variabilele din instrucțiunea **NEXT** pot fi omise, în care caz instrucțiunea **NEXT** se referă la cea mai recentă instrucțiune **FOR**. Este, totuși, o idee bună de a include variabilele pentru a preîntîmpina

orice confuzie.

Dacă instrucţiunea **NEXT** este întîlnită înaintea instrucţiunii **FOR** corespunzătoare, va fi semnalată eroare de tip **Next without FOR**.

**Exemplu:**

```
10 J=10: K=30
20 FOR I=1 TO J STEP 2
30 PRINT I;
40 K=K+10
50 PRINT K
60 NEXT
RUN
1 40
3 50
5 60
7 70
9 80
```

În acest exemplu, bucla se va executa de 10 ori. Spre deosebire de alte versiuni BASIC (în care bucla se execută doar de 6 ori), valoarea finală a variabilei de buclă este atribuită înaintea valorii iniţiale.

```
10 I=5
20 FOR I=1 TO I+5
30 PRINT I;
40 NEXT
RUN
1 2 3 4 5 6 7 8 9 10
```

## FRE

**Funcţie**

**Scop:** Atribuie unei variabile numărul de octeţi neutilizaţi din spaţiul de date BASIC.

**Format:** *v=FRE(x)*
*V=FRE(x$)*

**Comentariu:** *x* şi *x$* sînt argumente false.

Avînd în vedere că în BASIC şirurile pot avea lungime variabilă, acestea pot fi manipulate dinamic. Din această cauză spaţiul de memorie alocat şirurilor poate deveni fragmentat, ceea ce are repercursiuni asupra performanţelor programului. Funcţia **FRE** de un argument şir oarecare produce o "curăţenie generală" înainte de a evalua numărul neutilizat de octeţi. În acest timp, BASIC-ul colectează toate datele utile şi eliberează spaţiul neutilizat de memorie folosit cîndva, în timpul rulării, pentru şiruri.

Zona de date este comprimată, astfel putîndu-se continua pînă în momentul depăşirii memoriei de lucru.

**Exemplu:**

**PRINT FRE(0)**
**22325**

Evident că rezultatul acestui exemplu poate diferi în timpul rulării pe calculatorul dumneavoastră.

## GET

Instrucţiune (fişiere)

**Scop:** Citeşte o înregistrare dintr-un fişier aleator într-un tampon de memorie aleator.

**Format:** *GET [#]nrfis[,număr]*

**Comentariu:** *nrfis* este numărul sub care a fost deschis fişierul.

*număr* este numărul înregistrării de citit, între 1 şi 16 megaocteţi. Dacă argumentul *număr* este omis, în tamponul de memorie se va citi următoarea înregistrare (după ultima instrucţiune GET).

Instrucţiunea **GET** poate fi utilizată de asemenea în cazul fişierelor de comunicaţii, în care caz argumentul *număr* reprezintă numărul de octeţi ce poate fi citit din memoria tampon de comunicaţii. Acest număr nu poate depăşii valoarea atribuită prin opţiunea LEN din instrucţiunea OPEN "COM...

**Exemplu:** Acest exemplu deschide fişierul "FISA" pentru acces aleator, cu cîmpurile definite în linia 20. Instrucţiunea **GET** de la linia 30 citeşte o ˆnregistrare într-un tampon de memorie de fişier. Linia 40 afişează info maţii din înregistrarea citită.

```
10 OPEN "A:FISA" AS #1
20 FIELD 1,30 AS NUME$, 30 AS ADR$, 35 AS
ORAS$
30 GET 1
40 PRINT NUME$, ADR$, ORAS$
```

## GET

Instrucţiune (Grafică)

**Scop:** Citeşte puncte dintr-o arie a ecranului.

**Format:** *GET(x1,y1)-(x2,y2),vector*

**Comentariu:** *x1,x2,y1,y2* sînt coordonate în forma relativă sau absolută.

*vector* este numele vectorului în care va fi păstrată informaţia. GET citeşte atributele punctelor dintr-un patrulater specificat într-un vector.

Patrulaterul are colţurile (x1,y1) şi (x2,y2).

Instrucţiunile GET şi PUT pot fi utilizate pentru animarea obiectelor în modul grafic. Aceste instrucţiuni pot fi gîndite ca nişte operaţii de "pompare a biţilor" spre (PUT) şi dinspre (GET) ecran. Matricea (vectorul) este utilizată pentru păstrarea imaginii, ea trebuind să fie numerică. Mărimea matricei în octeţi este dată de relaţia:

**4 + INT((x * bitiperpixel + 7)/8) * y**

unde x şi y sînt lungimea orizontală respectiv verticală a patrulaterului. Valoarea lui *bitiperpixel* este 2 în medie rezoluţie şi 1 în înaltă rezoluţie. Să presupunem că dorim să utilizăm instrucţiunea **GET** pentru a "prinde" o imagine de 10 pe 12 pixeli în medie rezoluţie. Numărul de octeţi necesar este 4+INT((10*2+7)/8)*12, sau 40 octeţi. Numărul de octeţi per element de matrice este:

- 2 pentru şirul întreg
- 4 pentru şirul simplă precizie
- 8 pentru şirul dublă precizie

De aici rezultă că în cazul nostru se poate folosi o matrice întreagă cu cel puţin 20 de elemente. Informaţia de pe ecran este memorată în vector după cum urmează:

1. 2 octeţi pentru dimensiunea lui x în biţi.
2. 2 octeţi pentru dimensiunea lui y în biţi.
3. datele.

De reţinut că întregii sînt memoraţi: octetul inferior primul, apoi octetul superior. Datele pentru fiecare rînd de puncte sînt aliniate la stînga în interiorul unui octet, astfel că, dacă este memorat mai puţin de un multiplu de 8 biţi, restul octetului este umplut cu zerouri. **GET** şi PUT lucrează sensibil mai rapid în medie rezoluţie dacă **x1 MOD 4** este egal cu zero şi în înaltă rezoluţie dacă **x1 MOD 8** este de asemenea egal cu zero.

**Exemplu:** Vezi instrucţiunea (grafică) PUT.

## GOSUB şi RETURN

Instrucţiuni

**Scop:** Execută salt şi întoarcere dintr-o subrutină.

**Format:** *GOSUB linie1*

*RETURN [linie2]*

**Comentariu:** *linie1* este numărul liniei de început a unei subrutine.

*linie2* este un număr opţional de linie la care se dă controlul după executarea subrutinei. Dacă *linie2*, se omite controlul programului este dat instrucţiunii imediat următoare celei mai recente instrucţiuni **GOSUB**.

O subrutină poate fi apelată de ori de cîte ori este nevoie într-un program, sau poate fi apelată din cadrul altei subrutine, această întreţesere

fiind limitată doar de memoria disponibilă.

**Exemplu:**

```
10 GOSUB 70
20 PRINT "TERMINAT SUBRUTINA 1"
30 GOSUB 90
40 PRINT "TERMINAT SUBRUTINA 2"
50 PRINT "TERMINAT TOTUL"
60 END
70 PRINT "** SUBRUTINA 1 **"
80 RETURN
90 PRINT "** SUBRUTINA 2 **"
100 RETURN
RUN
** SUBRUTINA 1 **
TERMINAT SUBRUTINA 1
** SUBRUTINA 2 **
TERMINAT SUBRUTINA 2
TERMINAT TOTUL
```

## GOTO

Instrucţiune

**Scop:** Execută salt necondiţionat din secvenţa normală de program la un număr de linie specificat.

**Format:** *GOTO linie*

**Comentariu:** *linie* este un număr de linie din program.

Dacă se specifică numărul liniei unei instrucţiuni executabile, este executată acea instrucţiune precum şi cele care urmează. Dacă numărul de linie se referă la o instrucţiune neexecutabilă (REM sau DATA), programul continuă cu prima instrucţiune executabilă care urmează.

Instrucţiunea **GOTO** poate fi utilizată în modul direct pentru a da execuţia programului la o linie dorită (de exemplu în timpul depanării unui program).

**Exemplu:** În acest exemplu instrucţiunea GOTO din linia 60 pune programul într-o buclă infinită, ce se termină în momentul în care nu mai pot fi citite date din instrucţiunea DATA.

```
10 DATA 5,7,12
20 READ R
30 PRINT "R = ";R,
40 A = 3.14*R^2
50 PRINT "ARIA = ";A
```

```
60 GOTO 10
RUN
R = 5          ARIA = 78.5
R = 7          ARIA = 153.86
R = 12         ARIA = 452.16
OUT OF DATA IN 20
```

## HEX$

Funcţie

**Scop:** Evaluează un şir ce reprezintă valoarea hexazecimală a argumentului zecimal.

**Format:** *v$ = HEX$(n)*

**Comentariu:** *n* este o expresie numerică cu valori cuprinse în intervalul -32768 la 65535. Dacă *n* este negativ este folosită forma complementului faţă de doi. Astfel HEX$(-n) este echivalent cu HEX$(65535-n).

**Exemplu:**

```
10 INPUT X
20 A$ = HEX$(X)
30 PRINT X " ZECIMAL INSEAMNA " A$ "
HEXAZECIMAL"
40 GOTO 10
RUN
? 32
32 ZECIMAL INSEAMNA 20 HEXAZECIMAL
? 1023
1023 ZECIMAL INSEAMNA 3FF HEXAZECIMAL
```

## IF

Instrucţiune

**Scop:** Permite luarea unei decizii după evaluarea unei expresii.

**Format:** *IF expresie [,]THEN clauză [ELSE clauză]*
*IF expresie [,]GOTO linie [[,]ELSE clauză]*

**Comentariu:** *expresie* poate fi orice expresie numerică.

*clauză* poate fi o instrucţiune sau o secvenţă de instrucţiuni BASIC (separate prin două puncte), sau un număr de linie pentru salt.

*linie* este numărul unei linii de program.

Dacă *expresia* este adevărată (nonzero) atunci se va executa clauza **THEN** sau **GOTO**.

Dacă rezultatul *expresiei* este fals (zero), clauza **THEN** sau **GOTO** este ignorată, executîndu-se clauza **ELSE**, dacă este prezentă. Execuţia

continuă cu următorul număr de linie ce conţine o instrucţiune executabilă.

Instrucţiunile **IF-THEN-ELSE** pot fi întrepătrunse. Întrepătrunderea este limitată doar de lungimea liniei. De exemplu:

```
IF X>Y THEN PRINT "MAI MARE" ELSE IF Y>X
THEN PRINT "MAI MIC" ELSE PRINT "EGAL"
```

este o instrucţiune validă. În cazul în care instrucţiunea nu conţine acelaşi număr de clauze **ELSE** şi **THEN**, fiecărei clauze **ELSE** i se opune cea mai apropiată clauză **THEN**, astfel:

```
IF A=B THEN IF B=C THEN PRINT "A=C" ELSE
PRINT "A< >C"
```

nu va afişa "A< >C" cînd "A< >B".

**Exemplu:** În exemplul următor dacă I este cuprins între 10 şi 20, este calculat DB şi se execută salt la linia 300. Dacă I nu este cuprins în acest domeniu se va afişa mesajul "Greseala !". De notat utilizarea a două instrucţiuni în clauza **THEN**:

```
100 IF (I>10) AND (I<20) THEN
DB=1988-I: GOTO 300 ELSE
PRINT "Greseala !"
```

În exemplul următor, în linia 20, tot ceea ce urmează după **THEN** este parte a clauzei. Aceasta înseamnă că instrucţiunea NEXT nu este executată pînă cînd N=I. Cînd se execută linia 20, N nu este egal cu I astfel că evaluarea **IF** dă rezultat fals. Deci, instrucţiunea NEXT nu va fi executată, iar programul se va termina la linia 30. Dacă se doreşte execuţia buclei pînă cînd N=I, instrucţiunea NEXT trebuie să apară sau pe o linie separată, sau anticipată de o clauza **ELSE**:

```
10 N=15
20 FOR I=1 TO 20: IF N=I THEN 40: NEXT
30 PRINT "N < > I": END
40 PRINT "N = I"
RUN
N < > I
```

## INKEY$

Variabilă

**Scop:** Citeşte un caracter de la tastatură.

**Format:** *v$ = INKEY$*

**Comentariu:** INKEY$ citeşte doar un singur caracter, chiar dacă în tamponul de memorie al tastaturii aşteaptă mai multe caractere. Valoarea rezultantă este un şir de zero, unu sau două caractere.

- un şir nul (lungime zero) înseamnă că nu a fost citit nici un caracter de la tastatură.
- un şir de lungime un caracter conţine caracterul actual citit de la tastatură.
- un şir de lungime două caractere indică un cod special extins.

Înainte de utilizarea acestui caracter într-o funcţie sau instrucţiune BASIC, este necesară atribuirea rezultatului la o variabilă şir.

**Exemplu:** Următoarea secţiune a unui program opreşte execuţia pînă cînd este apăsată o tastă:

```
100 PRINT "Apasati o tasta pentru continuare"
110 A$ = INKEY$: IF A$ = "" THEN 110
```

Următorul exemplu poate fi utilizat pentru testarea codului de două caractere rezultant:

```
10 K$ = INKEY$
20 IF LEN(K$) = 20 THEN K$ = RIGHT$(K$,1)
```

## INP

Funcţie

**Scop:** Evaluează un octet citit de la portul *n*.

**Format:** *v = INP(n)*

**Comentariu:** *n* trebuie să fie cuprins între 0 şi 65535.

INP este complementara funcţiei OUT

**Exemplu:**

```
100 A = INP(255)
```

## INPUT

Instrucţiune

**Scop:** Primeşte ihformaţii de la tastatură în timpul execuţiei programului.

**Format:** *INPUT[;]["mesaj";]variabilă[,variabilă]...*

**Comentariu:** "*mesaj*" este un şir constant care indică cerinţa intrării.

*variabilă* este numele unei variabile numerice, şir sau element al unei matrici ce va recepţiona informaţia.

În timpul execuţiei, cînd programul ajunge la o instrucţiune INPUT, se afişează pe ecran un semn de întrebare pentru a indica faptul că programul aşteaptă introducerea unor date. Dacă este inclus un "*mesaj*" se va afişa acel şir de caractere. Dacă mesajul este urmat de punct şi virgulă după mesaj se va tipări şi un semn de întrebare. Dacă mesajul este urmat de virgulă, semnul de întrebare nu se va mai tipări. De exemplu, în instrucţiunea **INPUT "Data nasterii",DN$**, mesajul va fi afişat fără să fie urmat de un semn de întrebare.

În cazul în care programul aşteaptă introducerea a mai mult de un element de la tastatură, elementele vor fi introduse cu virgulă între ele.

Tipul elementelor introduse trebuie să coincidă cu tipul declarat al variabilelor cărora li se vor atribui.

Dacă instrucţiunea INPUT este imediat urmată de punct şi virgulă, apăsînd tasta ENTER, acesta nu va executa un **carriage return/line feed**, ceea ce înseamnă că cursorul va rămîne pe aceeaşi linie.

**Exemplu:**

```
10 PI=3.14
20 INPUT "Introduceti raza ";R
30 A=PI*R^2
40 PRINT "Aria cercului este ";A
50 END
RUN
Introduceti raza ?7.4
Aria cercului este 171.9464
```

## INPUT #

Instrucţiune

**Scop:** Citeşte date de la un periferic secvenţial sau fişier şi le atribuie unor variabile din program.

**Format:** *INPUT #nrfis,variabilă[,variabilă]...*

**Comentariu:** *nrfis* este numărul folosit în momentul deschiderii fişierului pentru intrare.

*variabilă* este numele unei variabile ce va avea drept corespondent un element al fişierului.

Fişierul secvenţial poate fi pe disc sau casetă; el poate fi un şir de date secvenţial de la un adaptor de comunicaţie sau chiar tastatură **(KYBD:)**.

Tipul datelor din fişier trebuie să corespundă cu tipul variabilei declarate în program. Elementele fişierului trebuie să apară intocmai ca şi în cazul în care acestea ar fi fost citite cu instrucţiunea INPUT. În cazul valorilor numerice vor fi ignorate spaţiile libere precum şi "carriage return" şi "line feed". Numerele se termină cu un spaţiu, o virgulă, un "carriage return" sau "line feed".

## INPUT$

Funcţie

**Scop:** Evaluează un şir de *n* caractere citite de la claviatură sau de la fişier.

**Format:** *v$ = INPUT$(n[,#nrfis])*

**Comentariu:** *n* este numărul de caractere ce urmează a fi citit.

*nrfis* este numărul fişierului utilizat în instrucţiunea precedentă OPEN. Dacă se omite *nrfis* citirea se va face de la tastatură.

Dacă se foloseşte pentru intrare tastatura, caracterele corespunzătoare introduse, nu vor fi afişate pe ecran.

Funcţia **INPUT$** permite citirea caracterelor cu semnificaţie pentru editorul BASIC ca de exemplu Backspace (cod ASCII 8). Dacă se doreşte citirea acestor caractere se vor folosi INPUT$ sau INKEY$ (niciodată INPUT sau LINE INPUT).

Pentru fişierele de comunicaţie, funcţia INPUT$ este de preferat instrucţiunilor INPUT # sau LINE INPUT # pentru care toate caracterele pot fi semnificative în comunicaţie.

**Exemplu:** În exemplul următor se citeşte un singur caracter de la tastatură ca răspuns la mesaj:

```
100 PRINT "Apasati C pentru continuare sau S pentru
stop"
110 X$ = INPUT$(1)
120 IF X$ = "C" THEN GOTO 200
130 IF X$ = "S" THEN STOP ELSE GOTO 100
```

## INSTR

Funcţie

**Scop:** Caută prima apariţie a şirului *y$* în şirul *x$*, rezultatul funcţiei fiind poziţia în şir pe care se găseşte acesta. Parametrul opţional *n* reprezintă poziţia în şirul *x$* de la care va începe operaţia de căutare.

**Format:** *v = INSTR([n,]x$,y$)*

**Comentariu:** *n* este o expresie numerică cu valori cuprinse între 1 şi 255.

*x$,y$* pot fi orice variabile şir, expresii şir sau constante şir.

Dacă *n* este mai mare decît LEN(x$), dacă x$ este nul sau dacă y$ nu poate fi găsit, rezultatul funcţiei **INSTR** este *n* (sau 1 dacă *n* nu este specificat).

**Exemplu:** În exemplul următor se caută şirul "B" în şirul "ABCDEB". Cînd căutarea se face de la primul caracter al şirului, "B" va fi găsit pe poziţia 2; cînd căutarea începe de la poziţia 4, "B" va fi găsit la poziţia 6:

```
10 A$ = "ABCDEB"
20 B$ = "B"
30 PRINT INSTR(A$,B$); INSTR(4,A$,B$)
```

**RUN**
**4 6**

## INT

Funcţie

**Scop:** Evaluează cel mai mare întreg care este mai mic sau egal cu argumentul *x*.

**Format:** *v=INT(x)*

**Comentariu:** *x* poate fi orice expresie numerică.

Vezi de asemenea funcţiile FIX şi CINT (care evaluează la rîndul lor valori întregi).

**Exemplu:** Acest exemplu arată modul în care funcţia INT trunchează numerele pozitive, dar rotunjeşte "în sus" numerele negative (în direcţia negativă).

**PRINT INT(45.77)**
**45**

**PRINT INT(-2.89)**
**-3**

## IOCTL

Instrucţiune

**Scop:** Permite trimiterea unui şir de control spre un "driver" de periferic, după ce acesta a fost, în prealabil, deschis.

**Format:** *IOCTL [#]nrfis,sir*

**Comentariu:** *nrfis* este numărul fişierului pentru "driver"-ul de periferic.

*şir* este o expresie şir conţinînd datele de control.

Sistemul de manipulare a fişierelor de intrare/ieşire BASIC permite crearea şi instalarea unor programe de control a perifericelor (driver) proprii utilizatorului. Instrucţiunea IOCTL şi funcţia IOCTL$ trimit şi recepţionează date de control de la periferice.

Un şir de comandă **IOCTL** poate fi de pînă la 255 octeţi lungime. Sînt acceptate comenzi multiple în cadrul şirului, separate unele de altele prin punct şi virgulă:

**"LF;PL66;LW132"**

Posibilele comenzi sînt determinate de tipul "driver"-ului instalat.

**Exemplu:** În exemplul care urmează se modifică lungimea paginii la imprimantă:

**10 OPEN "LPT1" FOR OUTPUT AS #1**
**20 IOCTL #1,"PL60"**

## IOCTL$

Funcţie

**Scop:** Citeşte un şir de date de control de la un "driver" de periferic care a fost, anterior, deschis.

**Format:** *v$ = IOCTL$([#]nrfis)*

**Comentariu:** *nrfis* este numărul fişierului deschis către periferic.

Funcţia **IOCTL$** poate fi utilizată pentru a confirma succesul executării comenzii IOCTL.

**Exemplu** Exemplul următor verifică dacă un şir de control a fost recepţionat corect.

```
10 OPEN "COM" AS #1
20 IOCTL #1,"SW132;GW"
30 IF IOCTL$(1) = "132" THEN
PRINT "INSTALAT CORECT"
```

## KEY

Instrucţiune

**Scop:** Instalează sau afişează tastele programabile (notate cu F1...F10).

**Format:** *KEY ON*

*KEY OFF*

*KEY LIST*

*KEY n,x$*

*KEY n,CHR$(KBflag) si CHR$(scan code)*

**Comentariu: KEY ON** determină afişarea valorilor tastelor programabile pe linia 25. Cînd lăţimea ecranului este de 40 de caractere, se vor afişa 5 din cele 10 taste programabile. Cînd lăţimea este de 80, toate cele 10 taste vor fi afişate.

**KEY OFF** şterge afişarea acestor valori de pe linia 25, permiţînd utilizarea prin program şi a acestei linii.

**KEY LIST** listează valorile celor 10 taste programabile

**KEY n,x$** permite programarea acestor taste cu o secvenţa de caractere ce va fi reprodusă automat la apăsarea tastei respective.

*n* este numărul tastei programabile. Poate lua orice valoare între 1 şi 10.

*x$* este o expresie şir ce va fi atribuită tastei *n*

**Exemple:**

```
50 KEY ON
```

afişează conţinutul tastelor programabile pe linia 25.

```
10 KEY OFF
```

şterge afişarea acestor taste de pe linia 25, dar ele rămîn active.

**10 KEY 1,"FILES" + CHR$(13)**

atribuie tastei numărul 1 conţinutul şirului "FILES" + Enter.

## KEY(n)

Instrucţiune

**Scop:** Activează sau dezactivează tastele specifice într-un program BASIC.

**Format:** *KEY(n) ON*
*KEY(n) OFF*
*KEY(n) STOP*

**Comentariu:** *n* este o expresie numerică cuprinsă între 1 şi 20, după cum urmează:

**1-10** tastele programabile F1 la F10.
**11** Cusor sus.
**12** Cusor stînga.
**13** Cusor dreapta.
**14** Cusor jos.
**15-20** taste definite sub forma:
**KEY n,CHR$(KBflag) + CHR$(scan code).**

## KILL

Comandă

**Scop:** Şterge un fişier de pe disc. Comanda KILL din BASIC este similară comenzii ERASE din DOS.

**Format:** *KILL numefis*

**Comentariu:** *numefis* este o expresie şir care specifică numele fişierului.

Comanda **KILL** poate fi utilizată pentru toate tipurile de fişiere. Numele trebuie să conţină şi extensia dacă există. De exemplu, se poate salva un program BASIC cu comanda:

**SAVE "TEST"**

Limbajul adaugă automat extensia **.BAS** la numele fişierului după comanda SAVE, dar nu şi după comanda KILL. Dacă se doreşte ştergerea acestui fişier trebuie introdusă comanda:

**KILL "TEST.BAS"**

## LEFT$

Funcţie

**Scop:** Extrage cele mai din stînga *n* caractere din şirul *x$*.

**Format:** *v$ = LEFT$(x$,n)*

**Comentariu:** *x$* este orice expresie şir.

*n* este o expresie numerică ce trebuie să fie cuprinsă între 0 şi 255. Specifică numărul de caractere ce urmează a fi citite. Dacă *n* este mai mare sau egal cu LEN(x$), atunci se va extrage întregul şir *x$*. Dacă n = 0, va fi extras şirul nul (de lungime zero).

**Exemplu:**

```
10 A$ = "MULT/PUTIN"
20 B$ = LEFT$(A$,4)
30 PRINT B$
RUN
MULT
```

## LEN

Funcţie

**Scop:** Evaluează numărul de caractere din şirul *x$*.

**Format:** *v = LEN(x$)*

**Comentariu:** *x$* este orice expresie şir.

Atît caracterele neprintabile cît şi spaţiile sînt incluse în numărul total de caractere.

**Exemplu:**

```
10 X$ = "ABCD DCBA"
20 PRINT LEN(X$)
RUN
9
```

## LET

Instrucţiune

**Scop:** Atribuie unei variabile valoarea unei expresii.

**Format:** *[LET] varabilă = expresie*

**Comentariu:** *variabilă* este numele unei variabile sau al unui element vectorial.

*expresie* este expresia a carei valoare este atribuită *variabilei*. Tipul expresiei (şir sau numerice) trebuie să concorde cu cel al variabilei, în caz contrar semnalîndu-se eroare.

Cuvintul **LET** este opţional, ceea ce înseamnă că semnul de egalitate este suficient pentru a atribui unei variabile o expresie.

**Exemplu:**

```
10 LET DORI = 12
20 LET E = DORI + 2
30 LET FDANCE$ = "HORA"
```

Aceleaşi instrucţiuni mai pot fi scrise şi:

```
10 DORI=12
20 E=DORI+2
30 FDANCE$="HORA"
```

## LINE

Instrucţiune

**Scop:** Desenează o linie sau un dreptunghi pe ecran.

**Format:** *LINE [(x1,y1)]-(x2,y2)[,[color][,B[F]][,stil]]*

**Comentariu:** *(x1,y1),(x2,y2)* sînt coordonatele în forma relativă sau absolută.

*color* este o expresie întreagă. Alege atributul de culoare al modului ecran curent.

*stil* este un număr întreg de 16 biţi folosit pentru a pune puncte pe ecran. Opţiunea *stil* poate fi folosită pentru a desena o linie punctată. Deoarece *stil* are lungimea de 16 biţi, un model de linie punctată poate arăta în felul următor:

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

ceea ce înseamnă &HAAAA în notaţie hexazecimală.

*B* desenează un dreptunghi avînd colţurile opuse de coordonate (x1,y1) respectiv (x2,y2).

*F* umple dreptunghiul cu o culoare.

**Exemple:** Programul următor desenează o linie punctată:

```
10 SCREEN 1,0
20 LINE (0,0)-(319,199),,,&HAAAA
```

Pentru a folosi forma relativă de introducere a celei de-a doua perechi de coordonate faţă de prima pereche, se va introduce de exemplu:

```
LINE (100,100)-STEP (10,-20)
```

care va desena o linie de la (100,100) la (110,80).

Exemplul următor va desena dreptunghiuri aleatoare umplute cu culori aleatoare:

```
10 CLS
20 SCEEN 1,0: COLOR 0,0
30 LINE -(RND*319,RND*199),RND*2+1,BF
40 GOTO 30 'dreptunghiurile se vor suprapune
```

## LINE INPUT

Instrucţiune

**Scop:** Citeşte o linie întreagă (pînă la 255 de caractere) de la tastatură într-o variabilă şir, ignorînd delimitatorii.

**Format:** *LINE INPUT[;]["comentariu";] varsir*

**Comentariu:** "*comentariu*" este o constantă şir ce va fi afişată pe

ecran. Semnul întrebării nu este afişat, doar dacă acesta face parte din şirul comentariu.

*varsir* este numele variabilei şir sau al elementului matricei la care se va atribui linia.

Dacă **LINE INPUT** este urmat imediat de punct şi virgulă, atunci apăsînd tasta Enter nu se va produce deplasarea cursorului la începutul liniei următoare; cu alte cuvinte cursorul rămîne pe aceeaşi linie.

## LINE INPUT #

Instrucţiune

**Scop:** Citeşte o linie întreagă (pînă la 255 de caractere), ignorînd delimitatorii, de la un fişier secvenţial într-o variabilă şir.

**Format:** *LINE INPUT #nrfis,varsir*

**Comentariu:** *nrfis* este numărul sub care a fost deschis fişierul.

*varsir* este numele variabilei şir sau elementului matricial căruia i se atribuie linia.

**LINE INPUT #** citeşte toate caracterele din fişierul secvenţial pînă la un "carriage return".

**Exemplu:**

```
10 OPEN "LST" FOR OUTPUT AS #1
20 LINE INPUT "Adresa? ";C$
30 PRINT #1,C$
40 CLOSE 1
50 OPEN "LST" FOR INPUT AS #1
60 LINE INPUT #1,C$
70 PRINT C$
80 CLOSE 1
RUN
Adresa? Bdul MAGHERU, nr. 111
Bdul MAGHERU, nr. 111
```

## LIST

Comandă

**Scop:** Afişează programul curent din memorie pe ecran sau pe alt periferic specificat.

**Format:** *LIST [linie1][-[linie2]][,numefis]*

**Comentariu:** *linie1,linie2* reprezintă un număr de linie cuprins între 0 şi 65529. *linie1* este prima linie de listat, iar *linie2* este ultima. Se poate folosi un punct (.) pentru a înlocui oricare din liniile de mai sus cu linia de program curentă.

*numefis* este o expresie şir ce indică numele fişierului.

**Exemple:** Exemplul de mai jos listează tot programul pe ecran:

**LIST**

Următorul exemplu listează linia 35:

**LIST 35**

Acest exemplu listează toate liniile de la 100 la sfîrşitul programului către primul adaptor de comunicaţii cu 1200 bps (biţi pe secundă), fără paritate, 8 biţi de date, şi 1 bit de stop:

**LIST 100- ,"COM1:1200,N,8,1"**

Ultimul exemplu listează programul curent de la început şi pînă la linia 150:

**LIST -150**

## LLIST

Comandă

**Scop:** Tipăreşte la imprimantă o parte dintr-un program sau un program întreg.

**Format:** *LLIST [lin1][-[lin2]]*

**Comentariu:** Vezi comanda LIST.

## LOAD

Comanda

**Scop:** Încarcă un program de la un periferic specificat în memorie şi, opţional, dă comanda de rulare.

**Format:** *LOAD numefis[,R]*

**Comentariu:** *numefis* este o expresie şir care indică numele fişierului.

Comanda **LOAD** închide toate fişierele deschise anterior şi şterge toate variabilele şi toate liniile de program rezidente în memorie anterior comenzii de încărcare.

**LOAD numefis,R** este echivalentă comenzii **RUN numefis.**

**Exemple:** Comanda următoare încarcă şi rulează programul PROG2.BAS:

**LOAD "PROG2.BAS",R**

aceasta fiind echivalentă cu comanda:

**RUN "PROG2.BAS"**

## LOC

Funcţie

**Scop:** Evaluează poziţia curentă într-un fişier.

**Format:** *v=LOC(nrfis)*

**Comentariu:** *nrfis* este numărul fişierului utilizat la deschidere.

În cazul fişierelor aleatoare, **LOC** evaluează numărul ultimei înregistrări scrise sau citite din momentul deschiderii.

Pentru fişierele secvenţiale, **LOC** evaluează numărul de înregistrări scrise sau citite din momentul deschiderii fişierului (o înregistrare pentru fişierele secvenţiale înseamnă un bloc de date de 128 octeţi).

Pentru fişierele de comunicaţii, **LOC** evaluează numărul de caractere ce aşteaptă să fie citit din memoria tampon de intrare.

**Exemple:** Exemplul următor opreşte execuţia programului după ce se contorizează 50 de înregistrări:

**100 IF LOC(1) > 50 THEN STOP**

Următorul exemplu rescrie o înregistrare după citire:

**100 PUT #1,LOC(1)**

## LOCATE

Instrucţiune

**Scop:** Poziţionează cursorul pe ecranul activ. Parametrii opţionali activează sau dezactivează cursorul şi redefinesc dimensiunile acestuia.

**Format:** *LOCATE [rînd][,[col][,[cursor][,[start][,stop]]]]*

**Comentariu:** *rînd* este o expresie numerică cuprinsă între 1 şi 25. Indică numărul liniei de ecran unde va fi plasat cursorul.

*col* este o expresie numerică cuprinsă între 1 şi 40 sau între 1 şi 80, depinzînd de modul ecran curent. Indică numărul coloanei de ecran unde va fi plasat cursorul.

*cursor* este o valoare ce indică starea cursorului (vizibil sau nu). Zero (0) indică dezactivat, iar unu (1) cursor activ.

*start* este linia de start a cursorului. Este o expresie numerică cuprinsă între 0 şi 31.

*stop* este linia de stop a cursorului. Este o expresie numerică cuprinsă între 0 şi 31.

Opţiunile *cursor, start* si *stop* nu se aplică în modurile grafice.

**Exemplu:**

**10 LOCATE 5,1,1,0,7**

## LOF

Funcţie

**Scop:** Evaluează numărul de octeţi alocat unui fişier (lungimea fişierului).

**Format:** *v = LOF(nrfis)*

**Comentariu:** *nrfis* este numărul fişierului utilizat la deschidere.

## LOG

Funcţie

**Scop:** Evaluează logaritmul natural al argumentului **x**.

**Format:** *v=LOG(x)*

**Comentariu:** *x* este orice expresie numerică mai mare ca zero.

Logaritmul natural este logaritmul în baza **e**. Pentru calculul în dublă precizie, se va folosi comanda **/D** în linia de comandă BASIC.

**Exemple:** Primul exemplu calculează logaritmul expresiei 45/7:

```
PRINT LOG(45/7)
1.860752
```

Al doilea exemplu calculează logaritmul lui **e** şi $e^2$:

```
E=2.718282
? LOG(E)
1
? LOG(E*E)
2
```

## LPOS

Funcţie

**Scop:** Evaluează poziţia curentă a capului imprimantei (LPT1:) în timpul tipăririi.

**Format:** *v=LPOS(n)*

**Comentariu:** *n* este o expresie numerică ce indică imprimanta testată, după cum urmează:

**0 sau 1** LPT1:

**2** LPT2:

**3** LPT3:

**Exemplu:**

```
100 IF LPOS(0)>60 THEN LPRINT CHR$(13)
```

## LPRINT şi LPRINT USING

Instrucţiuni

**Scop:** Tipăresc date pe o imprimantă.

**Format:** *LPRINT [lista de expresii ][;]*

*LPRINT USING v$;lista de expresii[;]*

**Comentariu:** *lista de expresii* Este o listă de expresii şir şi/sau numerice care urmează a fi tipărită. Expresiile trebuie să fie separate prin virgulă sau punct şi virgulă.

*v$* este o constantă şir sau o variabilă care identifică formatul folosit la tipărire. Acesta este explicat în detaliu la instrucţiunea PRINT.

## LSET şi RSET

Instrucţiuni

**Scop:** Mută date într-o memorie tampon de fişier aleator în pregătirea execuţiei instrucţiunii PUT (pentru fişiere).

**Format:** *LSET varsir=x$*

*RSET varsir=x$*

**Comentariu:** *varsir* este numele unei variabile definite în instrucţiunea FIELD.

*x$* este o expresie şir utilizată pentru plasarea informaţiei în cîmpul identificat de *varsir*.

Dacă *x$* necesită mai puţini octeţi decît cei specificaţi pentru *varsir* în instrucţiunea FIELD, **LSET** aliniază la stînga şirul în cîmp, în timp ce **RSET** aliniază la dreapta.

## MERGE

Comandă

**Scop:** Uneşte linii de program dintr-un fişier ASCII, cu programul curent din memorie.

**Format:** *MERGE numefis*

**Comentariu:** *numefis* este o expresie şir care indică numele fişierului.

Fişierul este căutat după numele său. Dacă este găsit, liniile de program din fişierul respectiv sînt unite cu liniile din programul curent. Dacă în programul încărcat există linii cu acelaşi număr cu cel din memorie, liniile din fişier înlocuiesc pe cele deja existente în memorie.

**Exemplu:** Exemplul următor uneşte fişierul "PROG.OLD" aflat pe cititorul de disc "B:" cu programul rezident:

**MERGE "B:PROG.OLD"**

## MID$

Funcţie şi Instrucţiune

**Scop:** Evaluează partea dorită a unui şir de caractere dat. Cînd este folosit ca instrucţiune, înlocuieşte o porţiune a unui şir cu un alt şir.

**Format:**

Ca funcţie: *v$=MID$(x$,n[,m])*

Ca instrucţiune: *MID$(v$,n[,m])=y$*

**Comentariu:**

Pentru funcţie (**v$=MID$...**):

*x$* este orice expresie întreagă;

*n* este o expresie întreagă cuprinsă între 1 şi 255;

*m* este o expresie întreagă cuprinsă între 0 şi 255.

Funcţia evaluează, din şirul *x$*, un şir de lungime *m* caractere, începînd cu al *n*-lea caracter. Dacă se omite *m* sau dacă în dreapta se află mai puţin de *m* caractere începînd cu caracterul *n*, atunci se vor evalua cele mai din dreapta caractere din şirul *x$* începînd cu caracterul *n*.

Pentru instrucţiune (**MID$... = y$**):

*v$* este o variabilă şir în care se va opera înlocuirea.

*n* este o expresie întreagă cuprinsă între 1 şi 255.

*m* este o expresie întreagă cuprinsă între 0 şi 255.

*y$* este o expresie şir.

Caracterele din şirul *v$*, începînd din poziţia *n* sînt înlocuite cu caracterele din şirul *y$*. Parametrul opţional *m* se referă la numărul de caractere din şirul *y$* ce urmează a fi înlocuit. Dacă *m* este omis atunci tot şirul *y$* este înlocuit.

**Exemple:** Primul exemplu utilizează funcţia **MID$** pentru a selecta porţiunea din mijlocul şirului B$:

```
10 A$ = "BUNA"
20 B$ = "DIMINEATA ZIUA SEARA"
30 PRINT A$;MID$(B$,10,5)
RUN
BUNA ZIUA
```

Următorul exemplu ilustrează utilizarea instrucţiunii **MID$**:

```
10 FISA$ = "POPESCU - 97.39.66"
20 TEL.NOU$ = "94.15.26"
30 MID$(FISA$,11,8) = TEL.NOU$
40 PRINT FISA$
RUN
POPESCU - 94.15.26
```

## MKDIR

Comandă

**Scop:** Crează un director pe discul specificat.

**Format:** *MKDIR cale*

**Comentariu:** *cale* este o expresie şir de pînă la 63 de caractere, ce specifică noul director ce urmează a fi creat.

**Exemple:**

```
MKDIR "UTIL"
MKDIR "UTIL\MIRCEA"
MKDIR "UTIL\MIRCEA\VASILE"
MKDIR "INUTIL"
```

**CHDIR "INUTIL"**
**MKDIR "TOT"**
**MKDIR "PARTIAL"**
**MKDIR "INUTIL\TOT"**
**MKDIR "INUTIL\PARTIAL"**

Exemplele de mai sus au creat o structură arbore de tipul:

```
                 ROOT
                /    \
            UTIL    INUTIL
            /        /    \
     MIRCEA       TOT    PARTIAL
       /
   VASILE
```

## MKI$,MKS$,MKD$

Funcţii

**Scop:** Converteşte valori de tip numeric în valori de tip şir.

**Format:** *v$ = MKI$(expresie întreagă)*
*v$ = MKS$(expresie simplă precizie)*
*v$ = MKD$(expresie dublă precizie)*

**Comentariu:** Orice valoare numerică ce urmează a fi plasată într-o memorie tampon de fişier aleator cu instrucţiunile LSET sau RSET trebuie să fie convertite într-un şir. MKI$ converteşte un întreg într-un şir de 2 octeţi, MKS$ converteşte un număr în simplă precizie într-un şir de 4 octeţi, iar MKD$ converteşte un număr în dublă precizie într-un şir de 8 octeţi.

Aceste funcţii diferă de funcţia STR$ deoarece ele nu schimbă în mod real valoarea lor prin conversia în caractere, ci doar prin modul în care limbajul interpretează acesti octeţi.

**Exemplu:**

```
100 FIELD #1, 4 AS D$, 20 AS N$
110 LSET D$ = MKS$(J)
120 LSET N$ = A$
130 PUT #1
```

## NAME

Comandă

**Scop:** Schimbă numele unui fişier de pe disc. Comanda NAME în BASIC este echivalentă comenzii RENAME din sistemul de operare DOS.

**Format:** *NAME numefis1 AS numefis2*

**Comentariu:** *numefis1* este numele fişierului ce va fi schimbat. *numefis2* este noul nume de fişier.

**Exemplu:**

**NAME "PROG.OLD" AS "PROG.NEW"**

## NEW

Comandă

**Scop:** Şterge programul curent din memorie precum şi toate variabilele.

**Format:** *NEW*

**Comentariu:** Comanda **NEW** este utilizată de obicei pentru a "îndepărta" un program din memorie înainte de a introduce un altul.

## OCT$

Funcţie

**Scop:** Evaluează un şir ce reprezintă valoarea octală a argumentului zecimal.

**Format:** *v$ = OCT$(n)*

**Comentariu:** *n* este o expresie numerică cu valori între -32768 şi 65535.

Dacă *n* este negativ, este utilizată forma complementului faţa de doi, ceea ce înseamnă că **OCT$(-n)** este identic cu **OCT$(65535-n)**.

**Exemplu:** Exemplul de mai jos transformă în octal numărul 24 zecimal:

**? OCT$(24)**
**30**

## ON COM(n)

Instrucţiune

**Scop:** Indică un număr de linie pentru salt condiţionat în momentul în care sosesc informaţii în memoria tampon de comunicaţii.

**Format:** *ON COM(n) GOSUB linie*

**Comentariu:** *n* este numărul adaptorului de comunicaţie (1 sau 2). *linie* este numărul liniei la care se execută saltul.

Instrucţiunea COM(n) ON trebuie executată pentru activarea adaptorului de comunicaţii *n*. Dupa COM(n) ON, dacă în instrucţiunea **ON COM(n)** se specifică un număr de linie diferit de zero, atunci, de fiecare dată cînd programul începe o nouă instrucţiune, limbajul verifică dacă au sosit informaţii pe adaptorul de comunicaţii. În caz afirmativ se execută un salt (GOSUB) la linia respectivă.

**Exemplu:**

```
150 ON COM(1) GOSUB 500
160 COM(1) ON
.
.
.
500 'dupa sosirea informatiei
.
.
.
590 RETURN 300
```

## ON ERROR

Instrucţiune

**Scop:** Permite detectarea erorilor şi specifică linia de început a subrutinei de manipulare a erorilor.

**Format:** *ON ERROR GOTO LINIE*

**Comentariu:** *linie* este numărul primei linii din rutina de manipulare a erorii.

Pentru a înceta detecţia erorilor. se execută instrucţiunea **ON ERROR GOTO 0.**

**Exemplu:** Exemplul de mai jos testează dacă fereastra cititorului de disc curent este deschisă:

```
10 ON ERROR GOTO 100
20 OPEN "DATA" FOR INPUT AS #1
30 END
.
.
.
100 IF ERR = 71 THEN LOCATE 23,1
    PRINT "DISK NOT READY"
110 RESUME NEXT
```

## ON-GOSUB si ON-GOTO

Instrucţiuni

**Scop:** Execută salt la una sau mai multe linii de program în funcţie de valoarea unei expresii.

**Format:** *ON n GOTO linie[,linie]...*
*ON n GOSUB linie,[linie]...*

**Comentariu:** *n* este o expresie numerică, rotunjită la un întreg, dacă este necesar. Este cuprinsă între 0 şi 255, în caz contrar semnalîndu-se

eroare.

*linie* este numărul liniei la care se execută saltul.

Valoarea lui *n* determină la care linie de program se va executa saltul. Dacă, de exemplu, valoarea lui *n* este 3, înseamnă că al 3-lea număr de linie de program din listă este cel la care se va face saltul.

Dacă valoarea lui *n* este 0 sau mai mare decît numărul de linii din listă (dar mai mic sau egal cu 255), se continuă execuţia programului cu următoarea instrucţiune din program.

**Exemplu:**

```
100 REM meniu
110 PRINT "1. Rutina 1"
120 PRINT "2. Rutina 2"
130 PRINT "3. Rutina 3"
140 PRINT "4. Rutina 4"
150 INPUT "Alegeti optiunea";C
160 ON C GOSUB 200,300,400,500
170 GOTO 100 'dupa executia rutinelor
        din nou la meniu
200 REM incepe prima subrutina
.
.
.
290 RETURN
300 REM incepe a doua subrutina
.
.
.
```

## ON KEY(n)

Instrucţiune

**Scop:** Atribuie un număr de linie cînd este detectată apăsarea pe o tastă programabilă sau de control a cursorului.

**Format:** *ON KEY(n) GOSUB linie*

**Comentariu:** *n* este o expresie numerică cuprinsă între 1 şi 20 care indică tasta ce urmează a fi detectată după cum urmează:

| | |
|---|---|
| **1-10** | tastele programabile F1 la F10 |
| **11** | cursor sus |
| **12** | cursor stînga |
| **13** | cursor dreapta |
| **14** | cursor jos |

**15-20** taste definite sub forma:
**KEY n,CHR$(KBflag) + CHR$(scan code).**

*linie* este numărul liniei de program unde se execută saltul.

**Exemplu:**

```
10 KEY 15,CHR$(&H04) + CHR$(70) 'Detecteaza
Ctrl-Break
20 KEY 16,CHR$(&H04 + &H08) + CHR$(83)
    'Detecteaza Ctrl-Alt-Del
30 ON KEY(15) GOSUB 1000
40 ON KEY(16) GOSUB 2000
50 KEY(15) ON: KEY(16) ON
.
.
.
1000 PRINT "Detectare Ctrl-Break"
1010 RETURN
2000 T = T + 1
2010 ON T GOTO 2100,2200,2300,2400,2500
2100 PRINT "PAS 1 REINITIALIZARE SISTEM":
RETURN
2200 PRINT "PAS 2 REINITIALIZARE SISTEM":
RETURN
2300 PRINT "PAS 3 REINITIALIZARE SISTEM":
RETURN
2400 PRINT "PAS 4 REINITIALIZARE SISTEM":
RETURN
2500 KEY(16) OFF 'Dezafecteaza detectia
    reinitializarii sistemului
```

## ON PEN

Instrucţiune

**Scop:** Atribuie un număr de linie de program pentru a transfera controlul în momentul activării creionului luminos (**light pen**).

**Format:** *ON PEN GOSUB linie*

**Comentariu:** *linie* este numărul liniei de program la care se dă controlul în momentul în care se detectează activarea creionului luminos. Dacă *linie* este egal cu 0 se încetează detecţia.

Pentru activarea acestei instrucţiuni trebuie folosită instrucţiunea PEN ON. După executarea instrucţiunii PEN OFF nu se mai verifică

prezența creionului luminos.

**Exemplu:**

```
10 ON PEN GOSUB 500
20 PEN ON
.
.
.
500 'Subrutina pentru creion luminos
.
.
.
599 RETURN
```

## ON PLAY(n)

Instrucțiune

**Scop:** Generează continuu muzică în modul fundal (**background**) în timpul execuției programului.

**Format:** *ON PLAY(n) GOSUB linie*

**Comentariu:** *n* este o expresie întreagă cuprinsă între 1 și 32 indicînd notele care trebuie detectate.

*linie* este numărul liniei subrutinei de detecție pentru instrucțiunea PLAY. Valoarea 0 oprește detectarea.

Pentru activarea acestei instrucțiuni trebuie folosită instrucțiunea PLAY ON.

**Exemplu:**

```
10 ON PLAY(5) GOSUB 500
20 PLAY ON
.
.
.
500 'Subrutina pentru muzica in mod fundal
.
.
.
599 RETURN
```

## ON STRIG(n)

Instrucțiune

**Scop:** Atribuie un număr de linie de program pentru detectarea apăsării butoanelor de la "joystick".

**Format:** *ON STRIG(n) GOSUB linie*

**Comentariu:** *n* poate lua valorile 0,2,4 sau 6 şi indică butonul apăsat, după cum urmează:

**0** butonul A1

**2** butonul B1

**4** butonul A2

**6** butonul B2

*linie* este numărul liniei la care se execută saltul. Valoarea 0 incetează detectarea apăsarii butoanelor de la joystick.

Pentru activarea acestei instrucţiuni trebuie folosită instrucţiunea STRIG(n) ON. După executarea instrucţiunii STRIG(n) OFF nu se mai verifică apăsarea butonului *n* de la joystick.

**Exemplu:**

```
10 ON STRIG(0) GOSUB 500
20 STRIG(0) ON
.
.
.
500 'Subrutina pentru primul buton de la joystick
.
.
.
599 RETURN
```

## ON TIMER

Instrucţiune

**Scop:** Transferă controlul unei linii de program după scurgerea unei perioade de timp definite.

**Format:** *ON TIMER(n) GOSUB linie*

**Comentariu:** *n* este o expresie numerică cuprinsă între 1 şi 86.400 (1 secundă la 24 ore).

*linie* este numărul liniei la care se execută saltul după scurgerea intervalului de timp precizat.

Pentru activarea acestei instrucţiuni trebuie folosită instrucţiunea TIMER ON. Încetarea detecţiei se face cu ajutorul instrucţiunii TIMER OFF.

**Exemplu:**

```
10 CLS
20 ON TIMER(60) GOSUB 10000
30 TIMER ON
```

```
.
.
.
10000 RIND=CSRLIN 'salveaza rindul curent
10010 COL=POS(0) 'salveaza coloana curenta
10020 LOCATE 1,1: PRINT TIME$
10030 LOCATE RIND,COL 'din nou acasa
10040 RETURN
```

## OPEN

Instrucţiune

**Scop:** Permite intrarea sau ieşirea către un fişier sau periferic.

**Format:** *OPEN numefis [FOR mod1] AS [#]nrfis [LEN=recl]*

sau: *OPEN mod2,[#]nrfis,numefis[,recl]*

**Comentariu:** *mod1* de forma:

**OUTPUT** specifică modul secvenţial de ieşire.

**INPUT** specifică modul secvenţial de intrare.

**APPEND** specifică modul secvenţial de ieşire, în care fişierul este poziţionat la sfîrşitul şirului de date în momentul deschiderii acestuia.

*mod2* este o expresie şir, primul caracter fiind unul din următoarele:

**O** specifică modul secvenţial de intrare.

**I** specifică modul secvenţial de ieşire.

**R** specifică modul aleator de intrare/ieşire.

Pentru ambele formate:

*nrfis* este o expresie întreagă care indică numărul fişierului deschis.

*numefis* este o expresie şir care indică numele fişierului.

*recl* este o expresie întreagă care, dacă este inclusă, indică lungimea înregistrării pentru fişierele aleatoare. Poate lua valori între 1 şi 32767. Dacă nu se specifică lungimea, se ia 128 octeţi.

**Exemplu:**

```
10 OPEN "DATA" FOR OUTPUT AS #1
10 OPEN "B:FISIER" AS 1 LEN=256
```

## OPEN "COM...

Instrucţiune

**Scop:** Deschide un fişier de comunicaţii. Este valid doar în prezenţa Adaptorului de comunicaţii asincrone.

**Format:** *OPEN "COMn:[viteză],[paritate][,data][,stop][,RS][,CS[n]][,DS[n]][,CD[n]][LF][,PE]" AS #nrfis LEN=număr*

**Comentariu:** *n* este 1 sau 2, indicînd numărul Adaptorului de comunicaţii asincrone.

*viteză* este rata de transmisie/recepţie în biţi pe secundă (bps). Vitezele acceptate sînt 75, 110, 150, 300, 600, 1200, 2400, 4800 şi 9600. Dacă nu este specificată, se consideră 300 bps.

*paritate* este o constantă de lungime un caracter şi indică tipul parităţii la transmisie sau recepţie, după cum urmează:

**S** SPACE: Bit de paritate transmis şi recepţionat ca un spaţiu (bit 0).

**O** ODD: Paritate de tip impară de transmitere sau/şi recepţie.

**E** EVEN: Paritate de tip pară de transmisie sau/şi recepţie.

**M** MARK: Biţii de paritate transmişi şi recepţionaţi marcaţi (1 bit).

**N** NONE: Nu se transmite şi nu se verifică paritatea la recepţie.

Dacă nu se specifică, se consideră EVEN (E).

*data* este o constantă întreagă ce indică numărul de biţi de date transmişi/recepţionaţi.

Valorile permise sînt: 5, 6, 7 sau 8. Dacă nu este specificată, se consideră 7.

*stop* este o constantă întreagă ce indică numărul de biţi de stop. Valorile permise sînt 1 sau 2.

*nrfis* este o expresie întreagă ce indică numărul fişierului de comunicaţie deschis.

*număr* este numărul maxim de octeţi ce poate fi citit din memoria tampon de comunicaţie. Dacă nu se specifică se consideră 128 octeţi.

Instrucţiunea **OPEN "COM...** alocă o memorie tampon pentru intrări/ieşiri la fel ca OPEN pentru fişierele de disc.

Opţiunile **RS, CS, DS, CD, LF** şi **PE** afectează semnalele de pe linia de comunicaţie după cum urmează:

**RS** suprimă RTS (Request To Send)

**CS[n]** controlează CTS (Clear To Send)

**DS[n]** controlează DSR (Data Set Ready)

**CD[n]** controlează CD (Carrier Detect)

**LF** trimite un "line feed" după fiecare "carriage return"

**PE** activează verificarea parităţii

Argumentul *n* din opţiunile **CS, DS** si **CD** specifică numărul de milisecunde de aşteptare a semnalului înainte de a indica eroare "**Device timeout**". *n* poate lua o valoare cuprinsă între 0 şi 65535. Dacă *n* este omis sau este egal cu zero nu se mai execută verificarea.

**Exemplu:**

```
      10 OPEN "COM1:" AS #1
sau   10 OPEN "COM1:9600,N,8,,CS,DS,CD" AS #1
```

## OPTION BASE

Instrucţiune

**Scop:** Declară valoarea minimă a elemenelor unei matrici.

**Format:** *OPTION BASE n*

**Comentariu:** *n* este 1 sau 0.

Dacă nu se specifică, valoarea lui *n* este 0. După execuţie, instrucţiunea:

**OPTION BASE 1** face ca valoarea celui mai mic element al unei matrici să aibă indicele 1. De exemplu elementele matricii A(5) vor fi: A(1)....A(5)

## OUT

Instrucţiune

**Scop:** Trimite un octet la unul din porturile de ieşire al calculatorului.

**Format:** *OUT n,m*

**Comentariu:** *n* este o expresie numerică indicînd numărul portului. Este cuprinsă între 0 şi 65535.

*m* este o expresie numerică indicînd valoarea ce urmează a fi trimisă. Poate lua valori între 0 şi 255.

Instrucţiunea **OUT** este complementara funcţiei INP.

**Exemplu:** Exemplul următor trimite valoarea 100 la portul de ieşire 32:

**100 OUT 32,100**

## PAINT

Instrucţiune

**Scop:** Umple o arie de ecran cu o culoare selectată. Are semnificaţie doar în modul grafic.

**Format:** *PAINT (x,y)[[,pic][,margine][,fundal]]*

**Comentariu:** *(x,y)* sînt coordonatele punctului din aria ce urmează a fi umplută. Coordonatele pot fi indicate în forma absolută sau relativă. Acest punct este utilizat ca punct de început.

*pic* poate fi o expresie numerică sau şir. Este utilizat pentru a umple aria respectivă cu o culoare sau un model. Dacă *pic* este un număr atunci umplerea se face cu unul din atributele de culoare permise. Dacă *pic* este o expresie şir atunci umplerea se face cu un model.

*margine* este o expresie numerică întreagă. Defineşte atributul de culoare pentru marginile figurii pictate.

*fundal* este o expresie şir de 1 octet utilizat în pictarea modelelor.

În rezoluţie medie se poate umple interiorul sau exteriorul unei arii de ecran definite cu una dintre cele patru culori din paleta curentă definită de instrucţiunea COLOR. Un exemplu este umplerea cu roşu a unui cerc verde sau înconjurarea unui cerc roşu cu verde.

Pentru pictarea modelelor, atributul *pic* trebuie să fie o expresie şir de forma:

**CHR$(&Hnn) + CHR$(&Hnn) + CHR$(&Hnn)...**

Secvenţa CHR$ specifică o mască de lungime 1 octet. În momentul afişării, masca specificată va lua locul culorii. Expresia şir poate conţine pînă la 64 octeţi:

| | | |
|---|---|---|
| | **x creşte ---->** | |
| | **7 6 5 4 3 2 1 0** | |
| 0,0 | x x x x x x x x x | Model 0 |
| 0,1 | x x x x x x x x x | Model 1 |
| 0,2 | x x x x x x x x x | Model 2 |
| . | | |
| . | | |
| . | | |
| 0,63 | x x x x x x x x x | Model 63 (maxim permis) |

Modelul ales este apoi repetat uniform peste aria definită de *margine*.

Următorul tabel indică valorile binare si hexazecimale asociate cu fiecare atribut în medie rezoluţie:

| **Paleta de culoare 0** | **Atibutul în binar** | **Modelul pentru desenarea unei linii în binar** | **Modelul pentru desenarea unei linii în hex.** |
|---|---|---|---|
| verde | 01 | 01010101 | &H55 |
| roşu | 10 | 10101010 | &HAA |
| maro | 11 | 11111111 | &HFF |

| **Paleta de culoare 1** | **Atibutul în binar** | **Modelul pentru desenarea unei linii în binar** | **Modelul pentru desenarea unei linii în hex.** |
|---|---|---|---|
| turquoise | 01 | 01010101 | &H55 |
| mov | 10 | 10101010 | &HAA |
| alb | 11 | 11111111 | &HFF |

**Exemplu:**

```
10 CLS: SCREEN 1,0: KEY OFF
20 MODEL$ = CHR$(&HAA) + CHR$(&HAA) +
      CHR$(&HAA) + CHR$(&H55) + CHR$(&H55) +
```

```
        CHR$(&HFF)
30 COLOR 0,0 'se alege paleta 0
40 VIEW (1,1)-(150,100),0,2
50 GOSUB 1000
60 COLOR 0,1 'se alege paleta 1
70 GOTO 1020
.
.
.
1000 PAINT (125,50),MODEL$,2
1010 RETURN
1020 GOTO 1020
```

## PEEK

Funcţie

**Scop:** Evaluează un octet citit de la o locaţie de memorie indicată.

**Format:** *v = PEEK(n)*

**Comentariu:** *n* este un întreg cu valori cuprinse între 0 şi 65535. *n* este offset-ul din segmentul curent de memorie definit de instrucţiunea DEF SEG.

Valoarea evaluată este un întreg cuprins între 0 şi 255.

Funcţia **PEEK** este complementară instrucţiunii POKE.

**Exemplu:** Următorul exemplu de program testează care adaptor de terminal este ataşat la sistem:

```
10 'testeaza tipul adaptorului
20 DEF SEG = 0
30 IF (PEEK(&H410) AND &H30) = &H30
    THEN IBMMONO = 1
    ELSE IBMMONO = 0
```

## PEN

Instrucţiune şi Funcţie

**Scop:** Citeşte creionul luminos.

**Format:**

Ca instrucţiune: *PEN ON*

*PEN OFF*

*PEN STOP*

ca funcţie: *v = PEN(n)*

**Comentariu:** Funcţia **v = PEN(n)** citeşte coordonatele creionului lu-

minos.

*n* este o expresie numerică cuprinsă între 0 şi 9 şi afectează valoarea evaluată de funcţie după cum urmează:

**0** Indică dacă creionul era activ în timpul ultimei validări. Atribuie variabilei *v* valoarea -1 dacă creionul era activ şi 0 în cealaltă situaţie.

**1** Evaluează coordonata x cînd creionul a fost ultima oară activ. Valoarea va fi cuprinsă între 0 şi 319 în medie rezoluţie şi 0 la 639 în rezoluţie înaltă.

**2** Evaluează coordonata y cînd creionul a fost ultima oară activ. Valoarea va fi cuprinsă între 0 şi 199.

**3** Evaluează valoarea curentă a creionului; activ -1, iar dezactivat 0.

**4** Evaluează ultima coordonată x validă cunoscută. Valoarea evaluată este cuprinsă între 0 şi 319 în rezoluţie medie şi respectiv 639 în rezoluţie înaltă.

**5** Evaluează ultima coordonată y validă cunoscută. Valoarea evaluată este cuprinsă între 0 şi 199.

**6** Evaluează poziţia rîndului de caractere unde creionul a fost ultima oară activat. Ia valori între 1 şi 24.

**7** Evaluează poziţia coloanei de caractere unde creionul a fost ultima oară activat. Cuprins între 1 şi 40 sau 80, funcţie de WIDTH.

**8** Atribuie ultimul rînd de caractere valid. Ia valori între 1 şi 24.

**9** Atribuie poziţia ultimei coloane de caractere valide. Ia valori între 1 şi 40 sau 80, funcţie de WIDTH.

Instrucţiunea **PEN ON** permite activarea funcţiei de citire **PEN**. După utilizarea creionului luminos se va executa instrucţiunea **PEN OFF**, ceea ce va creşte considerabil viteza de execuţie, în continuare, a programului.

**Exemplu:** Acest exemplu afişeaza valoarea creionului luminos de la ultima validare, precum şi valoarea curentă:

```
10 PEN ON
20 FOR I=1 TO 500
30 X=PEN(0): X1=PEN(3)
40 PRINT X,X1
50 NEXT
60 PEN OFF
```

## PLAY

Instrucţiune

**Scop:** Generează muzică specificată de un şir.

**Format:** *PLAY şir*

**Comentariu:** PLAY implementează un concept similar cu DRAW prin introducerea unui "limbaj de definiţii armonice".

*şir* este o expresie şir constînd dintr-o succesiune de comenzi muzicale.

Comenzile în instrucţiunea PLAY sînt:

**A** la **G** (opţional cu **#**, **+** sau **-** )

Cînta nota indicată în octava curentă. Semnul număr (**#**) sau semnul plus ( **+** ) indică un diez; semnul minus (**-**) indică un bemol. Semnele **#**, **+** şi **-** sînt permise doar dacă au corespondent în notaţia muzicală (de ex. **B#** este o notă invalidă).

**On** Octava. Atribuie octava curentă pentru notele ce urmează. Sînt 7 octave numerotate de la 0 la 6. Dacă nu se specifică, se atribuie octava 4.

**>n** Se deplasează o octavă în sus şi cînta nota *n*.

**<n** Se deplaseaza în jos o octavă şi cînta nota *n*.

**Nn** Cînta nota *n* care poate varia între 0 şi 84. În 7 octave posibile sînt 84 note, 0 însemnînd "pauză".

**Ln** Atribuie lungimea notei care urmează. Lungimea notei este **1/n**, unde *n* poate lua valori între 1 şi 64.

**Pn** Pauză. Lungimea pauzei este **1/n**, unde *n* poate lua valori între 1 şi 64.

**.** Punct. cînd este plasat după o nota face ca acea notă să fie cîntată ca notă accentuată.

**Tn** Tempo. Atribuie numărul de sferturi de notă pe minut. *n* poate varia între 32 şi 255. În lipsă se consideră 120.

**MF** Muzică solist. Muzica (creată prin SOUND sau PLAY) rulează în modul solist.

**MB** Muzică fundal. Muzica (creată prin SOUND sau PLAY) rulează în modul fundal. Fiecare sunet sau notă este plasată într-un tampon de memorie, permiţînd programului să ruleze în continuare în timp ce muzica este interpretată.

**MN** Muzică în modul normal. Fiecare notă este interpretată 7/8 din timpul specificat de L.

**ML** Muzică legato. Fiecare notă este interpretată la lungimea specificată de L.

**MS** Muzică stacato. Fiecare notă este interpretată 3/4 din lungimea specificată de L.

**Xvar;** Execută şirul specificat de *var*.

În toate aceste comenzi argumentul *n* poate fi o constantă (de ex. 6) sau **=var;** unde *var* este numele unei variabile. După numele variabilei

este necesar să apară punct şi virgulă (;). Spaţiile sînt ignorate.

**Exemplu:**

```
10 'octave
20 SC$ = "CDEFGAB"
30 PLAY "O0 XSC$;"
40 FOR I = 1 TO 6
50 PLAY " > XSC$;
60 NEXT
70 PLAY "O6 XSC$;"
80 FOR I = 1 TO 6
90 PLAY " < XSC$;"
100 NEXT
```

## PLAY(n)

Funcţie

**Scop:** Evaluează numărul de note curent din tamponul de memorie alocat pentru muzica de fundal.

**Format:** *v = PLAY(n)*

**Comentariu:** *n* este un argument fals ce poate lua orice valoare.

**PLAY(n)** are ca rezultat valoarea 0 dacă programul rulează în modul solist. Valoarea maximă pe care o poate evalua este 32, care este numărul de note maxim ce poate fi alocat în memoria tampon.

**Exemplu:**

```
10 PLAY "MB CDEFGAB"
20 IF PLAY(1) = 5 GOTO 1000
30 GOTO 2000
.
.
.
1000 PLAY "MB 04 T200 L4 MS GG#GE"
2000 END
```

## PMAP

Funcţie

**Scop:** Transformă coordonatele fizice în coordonate universale sau invers. Are semnificaţie doar în modul grafic.

**Format:** *v = PMAP(x,n)*

**Comentariu:** *x* coordonata punctului ce urmează a fi transformat.

*n* tipul transformării, după cum urmează:

**1** transformă coordonata universală *x* în coordonată fizică.

**2** transformă coordonata universală *y* în coordonată fizică.
**3** transformă coordonata fizică *x* în coordonată universală.
**4** transformă coordonata fizică *y* în coordonată universală.
Funcţia **PMAP** este utilizată pentru translatarea coordonatelor între sistemul universal definit în instrucţiunea WINDOW şi sistemul de coordonate fizice.

## POINT

Funcţie

**Scop:** Prima formă evaluează atributul unui punct de pe ecran. Cea de-a doua formă evaluează coordonatele grafice curente *x* şi *y*. Are semnificaţie numai în modul grafic.

**Format:** *v = POINT(x,y)*
*v = POINT(n)*

**Comentariu:** *(x,y)* sînt coordonatele punctului dorit. Ele trebuie date în forma absolută. Dacă punctul dat se află în afara domeniului, este evaluată valoarea -1. Valorile evaluate sînt 0, 1, 2 şi 3 în medie rezoluţie şi 0 sau 1 în rezoluţie înaltă.

*n* evaluează coordonatele grafice curente *x* şi *y*.
**0** evaluează coordonata fizică *x*.
**1** evaluează coordonata fizică *y*.
**2** evaluează coordonata universală *x* dacă instrucţiunea WINDOW este activă. În caz contrar evaluează coordonata fizică *x*.
**3** evaluează coordonata universală *y* dacă instrucţiunea WINDOW este activă. În caz contrar evaluează coordonata fizică *y*.

**Exemplu:** Acest exemplu ilustrează valoarea evaluată de către funcţia **POINT**:

```
10 CLS: SCREEN 1,0: KEY OFF
20 PRINT "POINT(n) cu WINDOW inactiv"
30 GOSUB 110
40 WINDOW (0,0)-(319,199)
50 PRINT "POINT(n) cu WINDOW activ"
60 GOSUB 110
70 PRINT "POINT(n) cu WINDOW si SCREEN
active"
80 WINDOW SCREEN (0,0)-(319,199)
90 GOSUB 110
100 END
110 PSET (5,15)
```

```
120 FOR I = 1 TO 3
130 PRINT POINT(I);
140 NEXT
150 PRINT: PRINT
160 RETURN
RUN
POINT(n) cu WINDOW inactiv
5 15 5 15
POINT(n) cu WINDOW activ
5 184 5 15
POINT(n) cu WINDOW si SCREEN active
5 15 5 15
```

## POKE

Instrucţiune

**Scop:** Scrie un octet într-o locaţie de memorie.

**Format:** *POKE n,m*

**Comentariu:** *n* trebuie să fie cuprins între 0 şi 65535. Indică offset-ul în segmentul curent unde datele urmează a fi scrise. Segmentul curent este definit de instrucţiunea DEF SEG.

*m* este valoarea ce urmează a fi scrisă în locaţia de memorie. Poate lua valori între 0 şi 255.

**ATENTIE !**

*Limbajul BASIC nu verifică offset-ul specificat. Va rămîne în grija programatorului să nu scrie date în zona stivei, a variabilelor sau a programului BASIC.*

## POS

Funcţie

**Scop:** Evaluează poziţia din rînd (numărul coloanei) a cursorului.

**Format:** *v = POS(n)*

**Comentariu:** *n* este un argument fals.

**Exemplu:** Exemplul următor mută cursorul pe linia următoare dacă acesta depăşeşte coloana 60:

```
IF POS(0) > 60 THEN PRINT CHR$(13)
```

## PRINT

Instrucţiune

**Scop:** Afişează date pe ecranul calculatorului.

**Format:** *PRINT [lista de expresii][;]*

sau ? *[lista de expresii][;]*

**Comentariu:** *lista de expresii* este o listă de expresii şir sau/şi numerice separate prin virgule, spaţii sau punct şi virgulă. Orice constante şir din listă trebuie închise între ghilimele.

Dacă lista este omisă se va afişa un rînd gol.

**Poziţiile de afişare**

Poziţia fiecărui termen afişat este determinată de punctuaţia folosită pentru separarea termenelor din listă. Limbajul divide linia în zone de afişare de cîte 14 coloane fiecare. În lista de expresii:

- tipărind o virgulă între expresii determină că următoarea valoare să fie afişată la începutul următoarei zone libere.
- tipărind un punct şi virgulă determină ca următoarea valoare să fie afişată imediat după ultima valoare.
- tipărind unul sau mai multe spaţii între expresii are acelaşi efect ca punct şi virgulă.

Dacă lista se termină cu o virgulă, un punct şi virgulă sau funcţiile TAB sau SPC, următoarea instrucţiune **PRINT** va începe afişarea pe aceeaşi linie.

**Exemple:**

```
10 X=5
20 PRINT X+5,X-5,X*(-5)
RUN
10            0            -25

10 INPUT X
20 PRINT X;" la patrat este";X^2;" si";
30 PRINT X;" la cub este";X^3
RUN
?9
9 la patrat este 81 si 9 la cub este 729
```

## PRINT USING

Instrucţiune

**Scop:** Afişează pe ecran şiruri sau numere utilizînd un format specificat.

**Format:** *PRINT USING v$;lista de expresii[;]*

**Comentariu:** *v$* este o constantă şir sau variabilă constînd din caractere speciale de formatare. Aceste caractere de formatare determină cîmpul şi formatul de afişare a şirurilor sau numerelor.

*lista de expresii* constă din expresii numerice sau şiruri ce urmează a fi afişate, separate prin virgulă sau punct şi virgulă.

**Cîmpuri şir**

Cînd instrucţiunea PRINT USING este folosită pentru afişarea şirurilor, poate fi folosit unul dintre următoarele formate:

! Specifică faptul că va fi afişat doar primul caracter din şirul dat.

**\n spatii** Specifică faptul că 2 + n caractere din şirul dat urmează a fi afişate.

**Exemplu:**

```
10 A$ = "ABC": D$ = "DEF"
20 PRINT USING "!"A$;D$;
30 PRINT USING "\ \";A$;D$
RUN
AD
ABCDEF
```

& Specifică un cîmp şir de lungime variabilă. În acest caz şirul va fi afişat exact cum este introdus.

**Exemplu:**

```
10 A$ = "ABC": D$ = "DEF"
20 PRINT USING "!";A$;
30 PRINT USING "&";B$
RUN
ADEF
```

**Cîmpuri numerice**

Cînd instrucţiunea PRINT USING este folosită pentru a afişa numere, se pot folosi următoa ele caractere speciale:

# Semnul număr este folosit pentru reprezentarea poziţiei fiecărei cifre. Dacă numărul de reprezentat are mai puţine cifre decît poziţiile specificate se execută automat alinierea la dreapta (precedat de spaţii) a numărului în cîmp. Punctul zecimal poate fi plasat în orice poziţie din cîmp. Dacă este necesar, numerele sînt rotunjite.

**Exemplu:**

```
PRINT USING "##.## ";10.2;5.3;66.789;.234
10.20     5.30    66.79   0.23
```

+ Semnul plus la începutul sau sfîrşitul formatului implică reprezentarea semnului numărului (plus sau minus) la începutul sau, respectiv, la sfîrşitul său.

**Exemplu:**

```
PRINT USING " + ##.## ";-68.95;15.33
-68.95   +15.33
PRINT USING "##.## + ";-68.95;15.33
68.95-  15.33+
```

- Semnul minus la sfîrşitul formatului implică reprezentarea numerelor negative cu semnul minus după numărul respectiv.

**Exemplu:**

**PRINT USING "##.##-";-68.95,22.449**
**68.95- 22.45**

****** Un asterisc dublu la începutul formatului face ca spaţiile din faţa cîmpului numeric să fie umplute cu asteriscuri. De asemenea semnul ** specifică poziţiile pentru încă două cifre.

**Exemplu:**

**PRINT USING "**#.# ";12.39;-0.9;765.4**
***12.4 *-0.9 765.4**

**$$** Semnul dolar dublu face ca semnul dolarului să fie afişat imediat în stînga numărului formatat. Semnul $$ specifică poziţiile a încă două cifre, dintre care una este a semnului dolar. Formatul exponenţial nu poate fi folosit cu semnul $$.

**Exemplu:**

**PRINT USING "$$###.## ";456.78;0.9;-765.1**
**$456.78 $0.90 -$765.10**

****$** Acest semn la începutul formatului combină efectele celor două simboluri de mai sus.

**Exemplu:**

**PRINT USING "**$##.##";2.34**
****$2.34**

, O virgulă plasată la stînga punctului zecimal afişează o virgulă la fiecare grup de trei cifre semnificative din stînga punctului zecimal. Virgula plasată la sfîrşitul formatului este tipărită ca parte a şirului.

**Exemple:**

**PRINT USING "####,.##";1234.5**
**1,234.50**

**PRINT USING "####.##,";1234.5**
**1234.50,**

^ ^ ^ ^ Acest semn plasat după format specifică reprezentarea în forma exponenţială.

**Exemple:**

**PRINT USING "##.##^ ^ ^ ^";234.56**
**2.35E02**

**PRINT USING ".##^ ^ ^ ^-";-88888**
**.889E05-**

PRINT USING "+.##^^^^";123
+.12E03

_ Semnul sublinierii face ca următorul caracter să fie afişat drept caracter literal.

**Exemplu:**

**PRINT USING "_!##.##_!";12.34**
**!12.34!**

Dacă numărul de afişat este mai mare decît cîmpul numeric specificat se va afişa automat semnul procentului în faţa numărului respectiv.

**Exemple:**

**PRINT USING "##.##";111.22**
**%111.22**

**PRINT USING ".##";0.999**
**%1.00**

## PRINT # şi PRINT # USING

Instrucţiuni

**Scop:** Scriu date secvenţiale într-un fişier.

**Format:** *PRINT #nrfis,[USING x$;] lista de expresii[;]*

**Comentariu:** *nrfis* este numărul fişierului deschis pentru ieşire.

*x$* este o expresie şir ce conţine caractere de formatare descrise la instrucţiunea PRINT USING.

*lista de expresii* este o listă de expresii şir sau/şi numerice ce urmează a fi scrise în fişier.

**Exemplu:**

```
10 A=123
20 B=6789
30 C=22.33
40 OPEN "DATA" FOR OUTPUT AS #1
50 PRINT #1,USING "$$###.##,";A;B;C
60 CLOSE
70 OPEN "DATA" FOR INPUT AS #1
80 INPUT #1,A$,B$,C$
90 CLOSE
100 PRINT A$,B$,C$
```

## PSET şi PRESET

Instrucţiuni

**Scop:** Desenează un punct la poziţia specificată de pe ecran. Are înţeles doar în modul grafic.

**Format:** *PSET (x,y)[,culoare]*
*PRESET (x,y)[,culoare]*

**Comentariu:** *(x,y)* sînt coordonatele punctului de afişat. Aceste coordonate pot fi date atît în forma absolută cît şi în forma relativă.

*culoare* este o expresie întreagă ce alege un atribut din domeniul de atribute de culoare a modului ecran curent.

**PRESET** este aproape identic cu **PSET**. Singura diferenţă constă în faptul că dacă nu este specificat parametrul *culore* în instrucţiunea PRESET, este selectat automat pentru fundal atributul (0). Dacă este inclus parametrul *culoare* cele două instrucţiuni sînt identice.

**Exemplu:**

```
10 CLS: SCREEN 1: KEY OFF
20 FOR I=0 TO 100
30 PSET (I,I)
40 NEXT
50 'sterge linia
60 FOR I=100 TO 0 STEP -1
70 PRESET (I,I)
80 NEXT
```

## PUT

Instrucţiune (Fişiere)

**Scop:** Scrie o înregistrare dintr-o zonă de memorie aleatoare într-un fişier aleator.

**Format:** *PUT [#]nrfis[,număr]*

**Comentariu:** *nrfis* este numărul sub care a fost deschis fişierul.

*număr* este numărul înregistrării ce urmează a fi scrisă, de la 1 octet la 16 Mocteţi.

Dacă parametrul *număr* este omis, înregistrarea va avea următorul număr disponibil de înregistrare.

Instrucţiunea **PUT** poate fi utilizată pentru un fişier de comunicaţii. În acest caz parametrul *număr* este numărul de octeţi ce urmează a fi scris în fişierul de comunicaţii. Acest număr trebuie să fie mai mic sau cel mult egal cu valoarea stabilită de opţiunea LEN din instrucţiunea OPEN "COM...

## PUT

Instrucţiune (Grafică)

**Scop:** Plotează imagini pe o arie specificată a ecranului. Are semnificaţie doar în modul grafic.

**Format:** *PUT (x,y),vector[,acţiune]*

**Comentariu:** *(x,y)* sînt coordonatele colţului din stînga sus a imaginii de transferat.

*vector* este numele unei matrici (vector) numerice conţinînd informaţia de transferat. Pentru mai multe informaţii în legatură cu acest vector vezi instrucţiunea GET (pentru grafică).

*acţiune* poate fi de tipul PSET, PRESET, XOR, OR sau AND. Dacă nu se specifică se consideră **XOR**.

Instrucţiunea PUT este opusă instrucţiunii GET în sensul că prima ia date din vector şi le plotează pe ecran.

**PSET** ia datele din matrice şi le transpune pe ecran.

**PRESET** este la fel cu **PSET** cu excepţia că este produsă imaginea complementară. De exemplu, în medie rezoluţie, care are un atribut maxim 3, atributul 0 al imaginii va determina plotarea lui cu atributul 3 şi viceversa, iar un atribut 1 va produce o imagine cu atributul 2 şi, evident, viceversa.

**AND, OR** şi **XOR** specifică operaţii logice pe biţii fiecărei imagini.

**AND** este utilizat în cazul în care există deja o imagine în aria de ecran pe care urmează a se face transferul.

**OR** este folosit pentru a suprapune imaginea de transferat peste o imagine existentă.

**XOR** este un mod special care poate fi folosit pentru animaţie. Aceasta permite mişcarea unei imagini fără alterarea fundalului.

În rezoluţie medie **AND, OR** şi **XOR** au următoarele efecte asupra culorii:

**AND**

| | val. vector |
|---|---|
| ecran | 0 1 2 3 |
| 0 | 0 0 0 0 |
| 1 | 0 1 0 1 |
| 2 | 0 0 2 2 |
| 3 | 0 1 2 3 |

**OR**

| | val. vector |
|---|---|
| ecran | 0 1 2 3 |
| 0 | 0 1 2 3 |
| 1 | 1 1 3 3 |
| 2 | 2 3 2 3 |
| 3 | 3 3 3 3 |

**XOR**

| | |
|---|---|
| | val. vector |
| ecran | 0 1 2 3 |
| 0 | 0 1 2 3 |
| 1 | 1 0 3 2 |
| 2 | 2 3 0 1 |
| 3 | 3 2 1 0 |

Animarea unui obiect poate fi realizată după cum urmează:

1. Se pune obiectul pe ecran cu instrucţiunea PUT (cu XOR);
2. Se recalculează noua poziţie a obiectului;
3. Se pune din nou obiectul pe ecran la vechea locaţie cu instrucţiunea PUT (cu XOR);
4. Se repetă pasul 1, de data aceasta punînd obiectul la noua locaţie.

Mişcările realizate în acest fel lasă fundalul neschimbat. Eventualele licăriri se pot reduce prin reducerea timpului între pasul 1 şi pasul 4 şi asigurînd un timp de întîrziere suficient între pasul 1 şi 3.

**Exemplu:** Acest exemplu arată cum poate fi mutat un cerc de-a lungul ecranului cu acţiunea XOR:

```
10 CLS: DEFINT A-Z: SCREEN 1: KEY OFF
20 DIM A(404)
30 CIRCLE (160,100),20,3
40 PAINT (160,100),2,3
50 GET (140,80)-(180,120),A: CLS
60 X=30: Y=50
70 FOR I=1 TO 20
80 PUT (X,Y),A,XOR
90 PUT (X,Y),A,XOR
100 X=X+10
110 NEXT
```

## RANDOMIZE

Instrucţiune

**Scop:** Iniţializează generatorul de numere aleatoare.

**Format:** *RANDOMIZE [n]*
*RANDOMIZE TIMER*

**Comentariu:** *n* este o expresie întreagă, simplă sau dublă precizie ce este utilizată ca "sămînţa" de număr aleator.

Dacă *n* este omis, limbajul BASIC suspendă execuţia şi cere o valoare cu mesajul:

**Random Number Seed (-32768 to 32767)?**

Dacă generatorul de numere aleatoare nu este iniţializat cu o valoare, funcţia RND va returna aceeaşi secvenţă de numere aleatoare de fiecare dată de cîte ori se începe rularea programului. Pentru a schimba secvenţa de numere aleatoare de cîte ori se începe rularea este necesară introducerea unei instrucţiuni **RANDOMIZE** la începutul programului şi schimbarea valorii iniţiale ("sămînţa") la fiecare rulare.

Ceasul intern poate fi folositor pentru a da "sămînţa" numărului aleator. Pentru a nu se mai afişa mesajul de mai sus este suficientă utilizarea funcţiei TIMER.

**Exemple:**

```
10 RANDOMIZE
20 FOR I = 1 TO 4
30 PRINT RND;
40 NEXT I
RUN
Random Number Seed (-32768 to 32767)?
```

Să presupunem că se răspunde cu 3. Programul continuă:

```
Random Number Seed (-32768 to 32767)? 3
.7655695 .3558607 .3742327 .1388798
RUN
Random Number Seed (-32768 to 32767)?
```

De data aceasta să presupunem că se răspunde cu 4. Programul continuă:

```
Random Number Seed (-32768 to 32767)? 4
.1719568 .5273236 .6879686 .713297
RUN
Random Number Seed (-32768 to 32767)?
```

Dacă se răspunde, din nou, cu 3, se va obţine aceeaşi secvenţă ca în cazul primei rulări:

```
Random Number Seed (-32768 to 32767)? 3
.7655695 .3558607 .3742327 .1388798
```

Următorul exemplu utilizează funcţia TIMER. De notat că de fiecare dată cînd se rulează programul se obţin secvenţe diferite de numere:

```
10 RANDOMIZE TIMER
20 FOR I = 1 TO 4
30 PRINT RND;
40 NEXT
RUN
.9590051 .1036786 .1464037 .7754918
```

```
RUN
.8261163 .3364552 .2005318 .3278248
```

## READ

Instrucţiune

**Scop:** Citeşte valori dintr-o instrucţiune DATA şi le atribuie unor variabile. Vezi instrucţiunea DATA.

**Format:** *READ var[,var]...*

**Comentariu:** *var* este o variabilă de tip numeric sau şir sau element al unei matrici care va recepţiona valorile citite din tabloul DATA.

Instrucţiunea **READ** trebuie folosită împreună cu instrucţiunea DATA.

O instrucţiune **READ** poate accesa mai multe instrucţiuni DATA, precum şi mai multe instrucţiuni **READ** pot accesa o instrucţiune DATA. Dacă numărul variabilelor din lista de variabile depăşeşte numărul elementelor din instrucţiunile DATA se va semnala eroare de tip "**Out of data**".

**Exemplu:**

```
10 FOR I = 1 TO 10
20 READ A(I)
30 NEXT I
40 DATA 3.08,4.23,5.74,9.02,3.15
50 DATA 6.19,8.34,9.29,3.68,5,92
```

## REM

Instrucţiune

**Scop:** Inserează comentarii într-un program.

**Format:** *REM comentariu*

**Comentariu:** *comentariu* poate fi orice secvenţa de caractere.

Instrucţiunile **REM** nu sînt executate, ci doar afişate în locul în care au fost inserate.

Comentariile pot fi adăugate prin precedarea comentariului respectiv de o ghilimea ('). Dacă se doreşte inserarea unui comentariu pe o linie care conţine şi alte instrucţiuni este necesar ca acel comentariu să fie plasat la sfîrşitul liniei de program.

**Exemplu:**

```
10 REM calculeaza viteza medie
20 SUM = 0 'initializeaza suma
30 FOR I = 1 TO 20
40 SUM = SUM + V(I)
```

Linia 20 mai pote fi scrisă:

**20 SUM = 0: REM initializeaza suma**

## RENUM

Comandă

**Scop:** Renumerotează liniile de program.

**Format:** *RENUM [nrnou][,[nrvechi][,increment]]*

**Comentariu:** *nrnou* este primul număr de linie de utilizat în noua secvenţă.

*nrvechi* este numărul liniei din programul curent de unde începe renumerotarea. Dacă nu se specifică se ia prima linie din program.

*increment* este pasul de numerotare în noua secvenţă. Dacă nu se specifică se consideră 10.

**RENUM** realizează schimbarea tuturor numerelor de linii ce urmează instrucţiunilor ERL, ELSE, GOSUB, GOTO, ON...GOSUB, ON...GOTO, RESTORE, RESUME, RETURN şi THEN. Dacă într-una din aceste instrucţiuni apare un număr de linie inexistent, va fi semnalată eroare de tip "**Undefined line number xxxxx in yyyyy**".

Instrucţiunea **RENUM** nu poate fi folosită pentru schimbarea ordinii liniilor de program, sau pentru a crea linii de program cu numere mai mari decît 65529.

**Exemple:**

**RENUM 300,,50**

Acest exemplu renumerotează, de asemenea, întregul program. Prima linie a noului program va fi 300, iar incrementul între linii 50.

**RENUM 1000,900,20**

Exemplul de mai sus renumerotează liniile începînd cu linia 900, prima linie din noul program va fi 1000, iar incrementul între linii 20.

## RESET

Comandă

**Scop:** Închide toate fişierele de disc şi goleşte memoria tampon de sistem.

**Format:** *RESET*

**Comentariu:** Dacă toate fişierele deschise se află pe disc, **RESET** este identic cu CLOSE.

## RESTORE

Instrucţiune

**Scop:** Permite recitirea instrucţiunilor DATA de la o linie specificată.

**Format:** *RESTORE [linie]*

**Comentariu:** *linie* este numărul de linie a unei instrucţiuni DATA din program.

**Exemplu:**

```
10 READ A,B,C
20 RESTORE
30 READ D,E,F
40 DATA 57,68,79
50 PRINT A;B;C;D;E;F
RUN
57 68 79 57 68 79
```

## RESUME

Instrucţiune

**Scop:** Continuă programul după execuţia unei proceduri de recuperare a erorii.

**Format:** *RESUME [0]*
*RESUME NEXT*
*RESUME linie*

**Comentariu:**

**RESUME** sau **RESUME 0**

Execuţia reîncepe la instrucţiunea care a cauzat eroarea.

**RESUME NEXT**

Execuţia reîncepe la linia imediat următoare celei ce a cauzat eroarea.

**RESUME linie**

Execuţia reîncepe la numărul de linie specificat.

**Exemplu:**

```
10 ON ERROR GOTO 1000
.
.
.
1000 IF (ERR=230) AND (ERL=90) THEN PRINT
     "Incerca din nou": RESUME 80
```

## RETURN

Instrucţiune

**Scop:** Termină execuţia unei subrutine şi face trimiterea în programul principal. Vezi GOSUB şi RETURN.

**Format:** *RETUTN [linie]*

**Comentariu:** *linie* este numărul liniei de program la care se doreşte întoarcerea.

## RIGHT$

Funcţie

**Scop:** Atribuie şirului *v$* cele mai din dreapta *n* caractere din şirul *x$*.

**Format:** *v$ = RIGHT$(x$)*

**Comentariu:** *x$* este orice expresie şir.

*n* este o expresie întreagă ce specifică numărul de caractere ce urmează a fi citite.

**Exemplu:**

```
10 N$ = "ION POPESCU"
20 PRINT RIGHT$(N$,7)
RUN
POPESCU
```

## RMDIR

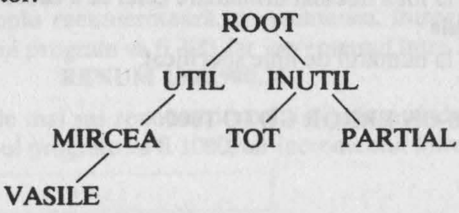
Comandă

**Scop:** Şterge un director de pe discul specificat.

**Format:** *RMDIR cale*

**Comentariu:** *cale* este o expresie şir ce identifică subdirectorul ce urmează a fi şters din directorul existent.

Înainte ca subdirectorul respectiv să poată fi şters trebuie ca acel subdirector să nu conţină nici un fişier. **Exemplu:** Să considerăm următorul arbore director:




Pentru a şterge subdirectorul "PARTIAL", se va executa:

**RMDIR "INUTIL\PARTIAL"**

## RND

Funcţie

**Scop:** Atribuie unei variabile un număr aleator cuprins între 0 şi 1.

**Format:** *v = RND[(x)]*

**Comentariu:** *x* este o expresie numerică ce afectează valoarea atribuită.

La fiecare rulare a programului este generată aceaşi secvenţă de numere aleatoare. Pentru a preîntîmpina acest lucru trebuie reiniţializat generatorul de numere aleatoare cu ajutorul instrucţiunii RANDOMIZE. Acelaşi lucru se poate face şi cu ajutorul funcţiei **RND** cu condiţia ca *x* să fie un număr negativ. Aceasta generează totdeauna o secvenţă particulară pentru un x dat. Secvenţa nu este afectată de către instrucţiunea RANDOMIZE; astfel, pentru a genera secvenţe diferite la fiecare rulare a programului, este necesar să se utilizeze diferite valori pentru *x*. Dacă *x* este pozitiv sau dacă lipseşte, va fi generat următorul număr aleator din secvenţa. **RND(0)** repetă ultimul număr generat. Pentru a avea numere aleatoare cuprinse între 0 şi n, se va folosi formula:

**INT(RND*(N+1))**

Exemplu:

```
10 FOR I=1 TO 3
20 PRINT RND(I);          'x>0
30 NEXT I
40 PRINT: X=RND(-6)  'x<0
50 FOR I=1 TO 3
60 PRINT RND(I);          'x>0
70 NEXT I
80 RANDOMIZE 853
90 PRINT X=RND(-6)   'x<0
100 FOR I=1 TO 3
110 PRINT RND;            'la fel ca x>0
120 NEXT I
130 PRINT: PRINT RND(0)
RUN
.6291626 .1948297 .6305799
.6818615 .4193624 .6215937
.6818615 .4193624 .6215937
.6215937
```

## RUN

Comandă

**Scop:** Începe execuţia unui program.

**Format:** *RUN [linie]*

*RUN numefis[,R]*

**Comentariu:** *linie* este numărul liniei din programul curent de la care se doreşte începerea execuţiei.

*numefis* este o expresie şir care indică numele fişierului de executat.

**RUN** sau **RUN linie** începe execuţia programului curent din memorie.

**RUN numefis** încarcă un fişier de pe disc în memorie şi începe execuţia lui. Înainte de a încărca programul specificat, această comandă închide toate fişierele deschise anterior şi şterge conţinutul memoriei, după care va executa încărcarea noului program în memorie. Dacă se adaugă opţiunea **R**, toate fişierele de date rămîn deschise.

**Exemple:**

```
10 PI=3.141593
20 PRINT PI
RUN
3.141593

RUN 20
0
```

## SAVE

Comandă

**Scop:** Salvează pe disc un fişier BASIC.

**Format:** *SAVE numefis[,A]*
*SAVE numefis[,P]*

**Comentariu:** *numefis* este o expresie şir care indică numele sub care va fi salvat programul curent din memorie pe disc.

Opţiunea **A** salvează programul în format ASCII. Dacă nu se specifică această opţiune, programul va fi salvat în format binar comprimat. Fişierele de tip ASCII ocupă puţin mai mult spaţiu pe disc, dar sînt cazuri cînd este necesară salvarea fişierelor de program sub acest format. Programele salvate în ASCII pot fi citite ca fişiere de date.

Opţiunea **P** (protejat) salvează programul într-un format binar codat. Un program protejat poate fi încărcat şi rulat, dar nu mai poate fi listat sau editat. Nu există modalităţi de "deprotejare" a unui program după ce acesta a fost în prealabil salvat cu opţiunea **P**.

**Exemple:**

```
SAVE "PROG1"
SAVE "FISIER",A
SAVE "SECRET.TOT",P
```

## SCREEN

Funcţie

**Scop:** Atribuie unei variabile codul ASCII (0-255) corespunzător caracterului din rîndul şi coloana specificată.

**Format:** *v = SCREEN(rînd,col,[z])*

**Comentariu:** *rînd* este o expresie numerică cu valori între 1 şi 25.

*col* este o expresie numerică cu valori între 1 şi 40 sau 1 şi 80.

*z* este o expresie numerică evaluată la adevărat sau fals.

În modul text, dacă se include parametrul opţional *z* şi acesta este adevărat (diferit de zero), atunci se va atribui variabilei atibutul de culoare al caracterului de la poziţia respectivă în locul codului caracterului respectiv. Atributul de culoare este un număr cuprins între 0 şi 255. Acest număr este:

**v MOD 16** atributul de culoare al caracterului;

**((v - c.car)/16) MOD 128** atributul de culoare pentru fundal, unde **c.car** este atributul de culoare al caracterului, calculat ca mai sus;

**V > 127** este adevărat (-1) dacă caracterul clipeşte şi fals (0) invers.

**Exemple:** Presupunînd că la poziţia (10,10) de pe ecran se află caracterul A, atunci:

```
10 X=SCREEN(10,10)
20 PRINT X
RUN
65
```

Următorul exemplu atribuie variabilei **x** valoarea atributului de culoare din partea stînga sus a ecranului:

```
100 X=SCREEN(1,1,1)
```

## SCREEN

Instrucţiune

**Scop:** Specifică atributul de culoare al ecranului pentru instrucţiunile ce urmează.

**Format:** *SCREEN [mod][,[culoare][,[apag][,vpag]]]*

**Comentariu:** *mod* este o expresie numerică care are ca rezultat o valoare întreagă, după cum urmează:

**0** modul text la lăţimea curentă (40 sau 80 de caractere);

**1** rezoluţie medie în modul grafic (320x200).

**2** rezoluţie înaltă în modul grafic (640x200).

*culoare* este o expresie numerică ce are ca rezultat o valoare adevărată sau falsă. Activează sau dezactivează culoarea. În modul text (mod = 0), o valoare falsă (zero) dezactivează culoarea (doar imaginile monocrome sînt afişate); o valoare adevărată (diferită de zero) activează culoarea (sint afişate imaginile color). În modul grafic, rezoluţie medie (mod = 1), o valoare adevărată (diferită de zero) va dezafecta culoarea, iar

o valoare falsă (zero) activează culoarea. În modul grafic, rezoluţie înaltă fiind doar două culori (alb şi negru), acest parametru nu are semnificaţie.

*apag* (pagina activă) selectează pagina ce urmează a fi afişată pe ecran; are valori cuprinse între 0 şi 7 pentru 40 de coloane şi 0 şi 3 pentru 80 de coloane. Este validă doar pentru modul text (mod = 0).

*vpag* (pagina vizuală) selectează pagina ce urmează a fi afişată pe ecran în acelaşi mod ca şi *apag* de mai sus. Pagina vizuală poate fi diferită de pagina activă. Ca şi *apag*, *vpag* este validă doar în modul text (mod = 0). Dacă este omisă *vpag* este egală cu *apag*.

Dacă toţi parametrii sînt corecţi, noul mod ecran este memorat; ecranul este curăţat; se va selecta culoarea alb pentru caracter şi negru pentru fundal şi margine.

## SGN

Funcţie

**Scop:** Atribuie unei variabile semnul lui *x*.

**Format:** *v = SGN(x).*

**Comentariu:** *x* este o expresie numerică oarecare.

SGN(x) este funcţia matematică signum:

- dacă *x* este pozitiv, **SGN(x)** va avea ca rezultat 1;
- dacă *x* este zero, **SGN(x)** va avea ca rezultat 0;
- dacă *x* este negativ, **SGN(x)** va avea ca rezultat -1.

**Exemplu:** Exemplul următor realizează salt la linia 100 dacă **x** este negativ; la 200 dacă **x** este zero şi la linia 300 dacă **x** este pozitiv:

**ON SGN(X) + 2 GOTO 100,200,300**

## SHELL

Instrucţiune

**Scop:** Încarcă şi execută un alt fişier de program (ca de exemplu fişiere cu extensie .COM, .BAT, .EXE). Orice program executat sub limbajul BASIC este considerat un proces descendent. La terminarea procesului descendent, controlul este preluat din nou de ascendentul lui BASIC, respectiv de instrucţiunea imediat următoare instrucţiunii **SHELL**.

**Format:** *SHELL [şir de comandă]*

**Comentariu:** *şir de comandă* este o expresie şir conţinînd numele programului de rulat şi, opţional, orice parametru ce trebuie trecut procesului descendent.

Procesul descendent este executat de instrucţiunea **SHELL** prin încărcarea şi rularea fişierului COMMAND.COM cu opţiunea /C. Dacă

instrucţiunea **SHELL** nu conţine **şirul de comandă**, se va încărca o copie a fişierului COMMAND.COM, va apare promptul DOS, iar din acest moment utilizatorul va putea introduce orice comandă validă în sistemul de operare DOS. Pentru întoarcerea la procesul ascendent BASIC se va introduce în linia de comandă DOS comanda **EXIT**.

**Exemple:**

```
10 OPEN "SORTIN.DAT" FOR OUTPUT AS #1
20 'scrie datele de sortat
.
.
.
100 CLOSE 1
110 SHELL "SORT < SORTIN.DAT >
SORTOUT.DAT"
120 OPEN "SORTOUT.DAT" FOR INPUT AS #1
130 'proceseaza datele sortate
.
.
.
```

Exemplul următor afişează directorul discului A: din BASIC:

```
SHELL
A>DIR (tipareste comanda dir sub promptul DOS)
.
.
.
A>EXIT (tipareste EXIT pentru reintoarcere în
BASIC)
```

Acelaşi rezultat se poate obţine mai simplu cu:

```
SHELL "DIR A:"
```

## SIN

Funcţie

**Scop:** Calculează funcţia sinus trigonometric.

**Format:** *v=SIN(x)*

**Comentariu:** *x* este unghiul în radiani.

Pentru conversia gradelor în radiani se va multiplica cu PI/180, unde PI=3.141593. Pentru calculul corect al acestei funcţii în dublă precizie se va specifica opţiunea **/D** în linia de comandă BASIC, în momentul iniţializării limbajului.

**Exemplu:** Considerînd că limbajul BASIC a fost iniţializat cu opţiu-

nea **/D** şi că dorim să calculăm sinus de 90 de grade, vom executa:

```
10 DEFDBL G,P,R
20 PI=ATN(1#)*4
30 GRADE=90
40 RADIANI=GRADE*PI/180 ' PI/2
50 PRINT SIN(RADIANI)
RUN
1
```

## SOUND

Instrucţiune

**Scop:** Generează sunete în difuzor.

**Format:** *SOUND frecv,durată*

**Comentariu:** *frecv* este frecvenţa dorită în Herţi (cicli pe secundă). Trebuie să fie o expresie numerică cu valori cuprinse între 37 şi 32767.

*durata* este durata dorită în eşantioane de timp. Într-o secundă sînt 18,2 eşantioane de timp. *durata* trebuie să fie o expresie numerică cu valori cuprinse între 0,0015 şi 65535.

După ce instrucţiunea **SOUND** generează un sunet, programul continuă execuţia pînă în momentul în care se ajunge la o nouă instrucţiune **SOUND**. Dacă *durata* noii instrucţiuni **SOUND** este 0, se opreşte execuţia instrucţiunii **SOUND** anterioare. Altfel, programul aşteaptă pînă cînd se termină execuţia primului sunet şi apoi dă execuţia celui de-al doilea.

| Nota | Frecventa | Nota | Frecventa |
|---|---|---|---|
| DO | 130.810 | DO | 523.250 |
| RE | 146.830 | RE | 587.330 |
| MI | 164.810 | MI | 659.260 |
| FA | 174.610 | FA | 698.460 |
| SOL | 196.000 | SOL | 783.990 |
| LA | 220.000 | LA | 880.000 |
| SI | 246.940 | SI | 987.770 |
| DO | 261.630 | DO | 1046.500 |
| RE | 293.660 | RE | 1174.700 |
| MI | 329.630 | MI | 1318.500 |
| FA | 349.230 | FA | 1396.900 |
| SOL | 392.000 | SOL | 1568.000 |
| LA | 440.000 | LA | 1760.000 |
| SI | 493.880 | SI | 1975.500 |

Durata unei măsuri poate fi calculată divizînd numărul de măsuri pe minut la 1092 (numărul de eşantioane de timp pe minut).

Tabelul următor indică tempo-urile tipice funcţie de numărul de eşantioane de timp:

| | Tempo | Măsuri/ Minut | Eşantioane/ Măsura |
|---|---|---|---|
| foarte lent | Larghissimo | | |
| | Largo | 40-60 | 27.3-18.2 |
| | Larghetto | 60-66 | 18.2-16.55 |
| | Grave | | |
| | Lento | | |
| | Adagio | 66-76 | 16.55-14.37 |
| lent | Adagietto | | |
| | Andante | 76-108 | 14.37-10.11 |
| mediu | Andantino | | |
| | Moderato | 108-120 | 10.11-9.1 |
| rapid | Allegretto | | |
| | Allegro | 120-168 | 9.1-6.5 |
| | Vivace | | |
| | Veloce | | |
| | Presto | 168-208 | 6.5-5.25 |
| foarte rapid | Prestissimo | | |

**Exemplu:** Următorul program crează un glisando:

```
10 FOR I=440 TO 1000 STEP 5
20 SOUND I,0.5
30 NEXT
40 FOR I=1000 TO 440 STEP -5
50 SOUND I,0.5
60 NEXT
```

## SPACE$

Instrucţiune

**Scop:** Are ca rezultat un şir de *n* spaţii.

**Format:** *v$ = SPACE$(n)*

**Comentariu:** *n* este cuprins între 0 şi 255.

**Exemplu:**

```
10 FOR I=1 TO 5
20 X$ = SPACE(I)
30 PRINT X$;I
40 NEXT I
RUN
```

```
1
 2
  3
   4
    5
```

## SPC

Funcţie

**Scop:** Sare *n* spaţii într-o instrucţiune PRINT.

**Format:** *PRINT SPC(n)*

**Comentariu:** *n* este o valoare cuprinsă între 0 şi 255.

Dacă *n* este mai mare decît lăţimea perifericului de afişare specificat, valoarea utilizată va fi **n MOD lăţime.**

**Exemplu:**

```
PRINT "A" SPC(15) "B"
A               B
```

## SQR

Funcţie

**Scop:** Atribuie unei variabile valoarea rădăcinii pătrate din *n*

**Format:** *v = SQR(x)*

**Comentariu:** *x* este o expresie numerică mai mare sau cel puţin egală cu 0.

Pentru calculul corect al acestei funcţii în dublă precizie se va specifica opţiunea /D în linia de comandă BASIC, în momentul iniţializării limbajului.

**Exemplu:**

```
10 FOR X = 10 TO 25 STEP 5
20 PRINT X,SQR(X)
30 NEXT
RUN
10          3.162278
15          3.872984
20          4.472136
25          5
```

## STICK

Funcţie

**Scop:** Atribuie unei variabile coordonatele x şi y a două joystick-uri.

**Format:** *v = STICK(n)*

**Comentariu:** *n* este o expresie numerică cu valori între 0 şi 3, ce afectează rezultatul după cum urmează:

**0** atribuie coordonata x pentru joystick-ul A.
**1** atribuie coordonata y pentru joystick-ul A.
**2** atribuie coordonata x pentru joystick-ul B.
**3** atribuie coordonata y pentru joystick-ul B.

**Exemplu:** Exemplul următor afişează 100 de eşantioane ale coordonatelor joystick-ului B:

```
10 PRINT "Joystick B"
20 PRINT "Coordonata x","Coordonata y"
30 FOR J=1 TO 100
40 TMP=STICK(0)
50 X=STICK(2): Y=STICK(3)
60 PRINT X,Y
70 NEXT
```

## STOP

Instrucţiune

**Scop:** Opreşte execuţia programului şi dă controlul la nivelul de comandă.

**Format:** *STOP*

**Comentariu:** Instrucţiunea **STOP** poate fi folosită oriunde în program pentru a întrerupe execuţia.

Spre deosebire de instrucţiunea END, instrucţiunea **STOP** nu închide fişierele.

Din nivelul de comandă se poate reîncepe execuţia programului de la instrucţiunea care urmează instrucţiunii **STOP** cu ajutorul comenzii CONT.

**Exemplu:**

```
10 INPUT A,B
20 TEMP=A*B
30 STOP
40 TFIN=TEMP+200: PRINT FINAL
RUN
? 26, 2.1
Break in 30

PRINT TEMP
54.6
CONT
254.6
```

## STR$

Funcţie

**Scop:** Transformă o valoare într-un şir de caractere ASCII.

**Format:** *v$ = STR$(x)*

**Comentariu:** *x* este orice expresie numerică.

Dacă *x* este pozitiv, şirul rezultat va conţine în faţă un spaţiu (spaţiul rezervat pentru semnul plus). De exemplu:

```
? STR$(321); LEN(STR$(321))
 321  4
```

Funcţia VAL este complementară funcţiei **STR$**.

## STRIG

Instrucţiune şi Funcţie

**Scop:** Citeşte statutul butoanelor de la joystick.

**Format:**

Ca instrucţiune: *STRIG ON*

*STRIG OFF*

Ca funcţie: *v = STRIG(n)*

**Comentariu:** *n* este o expresie numerică cu valori între 0 şi 7. Afectează valoarea atribuită după cum urmează:

**0** Are ca rezultat -1 dacă butonul A1 a fost apăsat de la ultima apelare a funcţiei STRIG(0); dacă nu, are ca rezultat 0.

**1** Are ca rezultat -1 dacă butonul A1 este în acel moment apăsat; dacă nu, are ca rezultat 0.

**2** Are ca rezultat -1 dacă butonul B1 a fost apăsat de la ultima apelare a funcţiei STRIG(2); dacă nu, are ca rezultat 0.

**3** Are ca rezultat -1 dacă butonul B1 este în acel moment apăsat; dacă nu, are ca rezultat 0.

**4** Are ca rezultat -1 dacă butonul A2 a fost apăsat de la ultima apelare a funcţiei STRIG(4); dacă nu, are ca rezultat 0.

**5** Are ca rezultat -1 dacă butonul A2 este în acel moment apăsat; dacă nu, are ca rezultat 0.

**6** Are ca rezultat -1 dacă butonul B2 a fost apăsat de la ultima apelare a funcţiei STRIG(6); dacă nu, are ca rezultat 0.

**7** Are ca rezultat -1 dacă butonul B2 este în acel moment apăsat; dacă nu, are ca rezultat 0.

Instrucţiunea **STRIG ON** trebuie executată înainte de a fi apelată funcţia **STRIG(n)**. După **STRIG(n)**, de fiecare dată cînd se începe o nouă instrucţiune, limbajul verifică dacă a fost apăsat un buton de la joystick. Pentru a dezafecta testul se apelează instrucţiunea **STRIG OFF**.

## STRIG(n)

Instrucţiune

**Scop:** Activează şi dezactivează testarea butoanelor de la joystick.

**Format:** *STRIG(n) ON*
*STRIG(n) OFF*
*STRIG(n) STOP*

**Comentariu:** **STRIG(n) ON** trebuie executată pentru a activa testarea butoanelor prin instrucţiunea STRIG(n)

Instrucţiunii **STRIG(n) OFF** dezafectează testarea.

Instrucţiunea **STRIG(n) STOP** dezafectează temporar testarea. În acest caz, dacă butonul este apăsat în acest interval, acţiunea este memorată şi, în momentul în care este activată instrucţiunea **STRIG(n) ON**, acţiunea respectivă este executată.

## STRING$

Funcţie

**Scop:** Atribuie unei variabile un şir de lungime *n* ale cărui caractere au codul ASCII *m* sau primul caracter al şirului *x$*.

**Format:** *v$ = STRING$(n,m)*
*v$ = STRING$(n,x$)*

**Comentariu:** *n,m* sînt cuprinse între 0 şi 255.

*x$* este orice expresie şir.

**Exemple:**

```
10 X$ = STRING$(10,45)
20 PRINT X$ "RAPORT LUNAR" X$
RUN
----------RAPORT LUNAR----------

10 X$ = "ABC"
20 S$ = STRING$(5,X$)
30 PRINT S$
RUN
AAAAA
```

## SWAP

Instrucţiune

**Scop:** Schimbă între ele valoarea a două variabile.

**Format:** *SWAP var1,var2*

**Comentariu:** *var1,var2* sînt numele a două variabile sau elemente ale unei matrici.

Operaţia de schimbare poate fi efectuată pe orice tip de variabilă (simplă precizie, dublă precizie sau şir), cu condiţia ca cele doua variabile să fie de acelaşi tip.

**Exemplu:**

```
10 A$ = " UNUL ": B$ = " TOTI ": C$ = "PENTRU"
20 PRINT A$,C$,B$
30 SWAP A$,B$
40 PRINT A$,C$,B$
RUN
UNUL PENTRU TOTI
TOTI PENTRU UNUL
```

## SYSTEM

Comandă

**Scop:** Iese din limbajul BASIC în sistemul de operare DOS.

**Format:** *SYSTEM*

**Comentariu:** Comanda **SYSTEM** închide toate fişierele înainte de întoarcerea în sistemul de operare. Dacă nu este salvat în prealabil, programul curent din memorie se pierde.

## TAB

Funcţie

**Scop:** Mută cursorul pe linia curentă în coloana *n*.

**Format:** *PRINT TAB(n)*

**Comentariu:** *n* este cuprins între 1 şi 255.

Funcţia **TAB** poate fi utilizată în instrucţiunile PRINT, LPRINT şi PRINT #.

**Exemplu:**

```
10 PRINT "NUME" TAB(25) "DATA NASTERII"
20 READ A$,B$
30 PRINT A$ TAB(25) B$
40 DATA "G. IONESCU","15 MAI 1930"
RUN
NUME                    DATA NASTERII
G. IONESCU              15 MAI 1930
```

## TAN

Funcţie

**Scop:** Atribuie unei variabile valoarea tangentei.

**Format:** $v = TAN(x)$

**Comentariu:** *x* este unghiul în radiani.

Pentru conversia gradelor în radiani se va multiplica cu PI/180, unde PI = 3.141593. Pentru calculul corect al acestei funcţii în dublă precizie se va specifica opţiunea **/D** în linia de comandă BASIC, în momentul iniţializării limbajului.

**Exemplu:** Acest program calculează tangenta de 45 grade:

```
10 PI=3.141593
20 GRADE=45
30 PRINT TAN(GRADE*PI/180)
RUN
1
```

## TIME

Variabilă şi Instrucţiune

**Scop:** Stabileşte sau citeşte timpul curent.

**Format:**

Ca variabilă: *v$ = TIME$*

Ca instrucţiune: *TIME$ = x$*

**Comentariu:**

**Pentru variabilă (v$ = TIME$):**

Timpul curent este citit ca un şir de 8 caractere. Şirul este de forma *hh:mm:ss*, unde *hh* sînt orele (0 la 23), *mm* sînt minutele (0 la 59), iar *ss* reprezintă secundele (0 la 59).

**Pentru instrucţiune (TIME$ = x$):**

Stabileşte timpul curent "potrivind" ceasul calculatorului. *x$* este o expresie şir (de forma *hh:mm:ss*) indicînd timpul ce urmează a fi stabilit.

**Exemplu:** Exemplul de mai jos afişează continuu timpul pe ecran:

```
10 CLS
20 LOCATE 10,15
30 PRINT TIME$
40 GOTO 20
```

## TIMER

Funcţie

**Scop:** Are ca rezultat un număr în simplă precizie reprezentînd numărul de secunde trecute de la miezul nopţii sau de la iniţializarea sistemului.

**Format:** *v = TIMER*

**Exemplu:**

```
10 TIME$="23:59:59"
```

```
20 FOR I=1 TO 70
30 PRINT "TIME$= ";TIME$;" TIMER=";TIMER
40 NEXT
RUN
TIME$= 23:59:59 TIMER= 86399.06
TIME$= 23:59:59 TIMER= 86399.11
TIME$= 23:59:59 TIMER= 86399.18
.
.
.
TIME$= 24:00:00 TIMER= 0
TIME$= 00:00:00 TIMER= .05
TIME$= 00:00:00 TIMER= .16
TIME$= 00:00:00 TIMER= .23
```

## TRON şi TROFF

Comenzi

**Scop:** Trasează execuţia instrucţiunilor din program.

**Format:** *TRON*
*TROFF*

**Comentariu:** Ca un ajutor în depanarea programelor, comanda **TRON** poate fi introdusă atît în interiorul programului cît şi în modul direct, activînd ca trasor numărul liniei de program în execuţie. Numerele liniilor de program apar în paranteze drepte. Trasarea se dezafectează cu comanda **TROFF**.

**Exemplu:**

```
10 K=10
20 FOR I=1 TO 2
30 L=K+10
40 PRINT J;K;L
50 K=K+10
60 NEXT
70 END
TRON

RUN
[10][20][30][40] 0 10 20
[50][60][30][40] 0 20 30
[50][60][70]
TROFF
```

## USR

Funcţie

**Scop:** Apelează subrutina în cod maşină indicată de argumentul *arg*.

**Format:** *v = USR[n](arg)*

**Comentariu:** *n* este un număr cuprins între 0 şi 9 corespunzînd numărului stabilit de instrucţiunea DEF USR pentru rutina dorită. Dacă se omite *n* se ia USR0.

*arg* este orice expresie numerică sau variabilă şir ce va trece ca argument în subrutina limbaj masină. Dacă subrutina nu cere argumente, este necesară prezenţa unui argument fals.

**Exemplu:**

```
10 DEFINT A-Z
20 OPTION BASE 1
30 X=0: Y=0: Z=0
40 DIM M(&H14)
50 Z=VARPTR(M(1))
60 BLOAD "SUBRT.COM",Z
70 DEF USR0=Z
80 X=5
90 Y=USR0(X)
100 PRINT Y
```

## VAL

Funcţie

**Scop:** Atribuie unei variabile valoarea numerică a şirului *x$*.

**Format:** *v = VAL(x$)*

**Comentariu:** *x$* este o expresie şir.

Funcţia **VAL** scoate toate spaţiile din şirul agument pentru a determina rezultatul. De exemplu:

**VAL("      -3")**

va avea ca rezultat -3.

Dacă primul caracter al şirului *x$* nu este numeric, rezultatul funcţiei **VAL(x$)** va fi 0 (zero).

## VARPTR

Funcţie

**Scop:** Are ca rezultat offset-ul segmentului curent de memorie a variabilei argument.

**Format:** *v = VARPTR(var)*

**Comentariu:** *var* este numele unei variabile şir sau numerice sau al unui element de matrice din program.

Valoarea rezultată este cuprinsă între 0 şi 65535. Acest număr este offset-ul din segmentul de date BASIC al primului octet de date identificat cu variabila.

Funcţia **VARPTR** este folosită pentru a obţine offset-ul unei variabile din segmentul de date BASIC, astfel că ea poate fi folosită pentru apelarea subrutinelor în limbaj maşină în scopul trecerii variabilelor spre acestea.

**Exemplu:**

```
10 DEFINT A-Z
20 DATA1 = 500
30 P = VARPTR(DATA1)
40 V = PEEK(P) + 256*PEEK(P + 1)
50 PRINT V
```

## VARPTR$

Funcţie

**Scop:** Atribuie unei variabile, sub formă de caractere, offset-ul unei variabile din memorie. Este folosită, de obicei, cu instrucţiunile PLAY si DRAW.

**Format:** *v$ = VARPTR$(var)*

**Comentariu:** *var* este numele unei variabile din program.

Se poate folosi **VARPTR$** pentru a indica numele unei variabile în şirul de comenzi ale instrucţiunilor PLAY sau DRAW. De exemplu:

**PLAY "XA$;"**

se mai poate scrie:

**PLAY "X" + VARPTR$(A$)**

sau:

**PLAY "O = I;"**

se mai poate scrie:

**PLAY "O = " + VARPTR$(I)**

## VIEW

Instrucţiune

**Scop:** Defineşte un subset rectangular din ecran la care face apel instrucţiunea WINDOW. Are inţeles doar în modul grafic.

**Format:** *VIEW [ [SCREEN][x1,y1)-(x2,y2)[,[culoare][,[margine]]]] ]*

**Comentariu:** *(x1,y1)-(x2,y2)* sînt coordonatele sus-stînga, respectiv

jos-dreapta ale ferestrei definite. x şi y trebuie să fie în limitele actuale definite ale ecranului.

*culoare* permite umplerea ferestrei cu o culoare. *culoare* este o expresie întreagă care poate fi unul dintre atributele de culoare permise.

*margine* permite încadrarea ferestrei cu o linie. *margine* este o expresie întreagă cu valori între limitele descrise la parametrul *culoare*.

Dacă argumentul SCREEN este omis, toate punctele plotate sînt relative la fereastră. Dacă se include argumentul SCREEN , toate punctele plotate sînt absolute şi pot fi atît în interiorul cît şi în exteriorul limitelor ecranului.

**VIEW** fără parametrii defineşte tot ecranul ca fereastră.

Prin această metodă se pot defini mai multe ferestre, dar numai una poate fi activă la un moment dat.

**Exemplu:** Următorul exemplu defineşte patru ferestre:

```
10 SCREEN 1: VIEW: CLS: KEY OFF
20 VIEW(1,1)-(151,91),,1
30 VIEW(165,1)-(315,91),,2
40 VIEW(1,105)-(151,195),,2
50 VIEW(165,105)-(315,195),,1
60 LOCATE 2,4:PRINT "Fereastra 1"
70 LOCATE 2,25: PRINT "Fereastra 2"
80 LOCATE 15,4: PRINT "Fereastra 3"
90 LOCATE 15,25: PRINT "Fereastra 4"
100 VIEW(1,1)-(151,91): GOSUB 1000
200 VIEW(165,1)-(315,91): GOSUB 2000
300 VIEW (1,105)-(151,195): GOSUB 3000
400 VIEW (165,105)-(315,195): GOSUB 4000
900 END
1000 CIRCLE (65,50),30,2
1010 ' Deseneaza un cerc în prima fereastra
1020 RETURN
2000 LINE (45,50)-(90,75),1,B
2010 ' Deseneaza un patrat in a doua fereastra
2020 RETURN
3000 FOR D = 0 TO 360: DRAW "ta = d;nu20": NEXT
3010 ' Deseneaza spite in a treia ferestra
3020 RETURN
4000 PSET (60,50),2: DRAW "e15 f15 l30"
4010 ' Deseneaza un triunghi in ultima fereastra
4020 RETURN
```

Următorul exemplu demonstrează scalarea cu ajutorul instrucţiunii **View:**

```
10 KEY OFF: CLS: SCREEN 1,0: COLOR 0,0
20 WINDOW SCREEN (320,0)-(0,200)
30 GOTO 80
40 C=1
50 CIRCLE (160,100),60,C,,,5/18
60 CIRCLE (160,100),60,C,,,1
70 RETURN
80 GOSUB 40: FOR I=1 TO 1500: NEXT I: CLS
90 VIEW (1,1)-(160,90),,2: GOSUB 40
100 GOTO 100
```

## WAIT

Instrucţiune

**Scop:** Suspendă execuţia programului în timpul monitorizării unuia dintre porturile calculatorului.

**Format:** *WAIT port,n[,m]*

**Comentariu:** *port* este numărul unuia dintre porturile calculatorului, între 0 şi 65535.

*n,m* sînt expresii întregi cu valori între 0 şi 255.

Instrucţiunea WAIT suspendă execuţia programului pînă la primirea de la portul dorit al calculatorului a unui şir de biţi specificat.

Datelor citite de la port li se aplică operaţia XOR cu expresia întreagă *m*, apoi operaţia AND cu *n*. Apoi se execută salt înapoi şi se citesc, din nou, datele de la port. Dacă rezultatul este diferit de zero, execuţia continuă cu următoarea instrucţiune. Dacă *m* se omite, se ia 0.

Instrucţiunea **WAIT** permite testarea a unuia sau mai mulţi biţi de la portul de intrare. Se pot testa poziţiile biţilor atît pentru 1 cît şi pentru 0. Poziţiile biţilor de testat sînt specificate punînd 1-uri pentru acele poziţii în *n*. Dacă nu se specifică *m*, biţii portului de intrare sînt testaţi pentru 1. Dacă *m* este specificat, un 1 în orice poziţie a lui *m* (pentru care există un bit de 1 în *n*) face ca **WAIT** să testeze biţii de 0 la intrare.

**Exemplu:** Acest exemplu suspendă execuţia programului pînă în momentul în care recepţionează un bit de 1 pe poziţia 2 de la portul 32:

```
WAIT 32,2
```

## WHILE si WEND

Instrucţiuni

**Scop:** Execută o serie de instrucţiuni într-o buclă atîta timp

cît o condiţie dată este adevărată.

**Format:**

**WHILE exp**

.

.

.

**(buclă de instrucţiuni)**

.

.

.

**WEND**

**Comentariu:** *exp* este orice expresie numerică.

Dacă *exp* este adevărată (diferită de zero), se execută bucla de instrucţiuni pînă cînd este înregistrată instrucţiunea **WEND**. Dacă *exp* este în continuare adevărată, procesul se repetă. Dacă *exp* devine falsă (egală cu zero), execuţia continuă cu instrucţiunea imediat următoare instrucţiunii **WEND**.

**Exemplu:** Exemplul de mai jos sortează elementele vectorului A în ordine alfabetică. Matricea A a fost definită cu J elemente.

```
100 F=1
110 WHILE F
120 F=0
130 FOR I=1 TO J-1
140 IF A(I)>A(I+1) THEN SWAP A(I),A(I+1):
F=1
150 NEXT I
160 WEND
```

## WIDTH

**Instrucţiune**

**Scop:** Fixează numărul de caractere pe o linie. După tipărirea numărului respectiv de caractere, limbajul aşează cursorul pe o linie nouă în prima coloană din stînga, prin adăugarea unui "carriage return".

**Format:** *WIDTH mărime*
*WIDTH periferic,mărime*
*WIDTH #nrfis,mărime*

**Comentariu:** *mărime* este o expresie numerică cuprinsă între 0 şi 255. Aceasta este noua lăţime. **WIDTH 0** are aceeaşi semnificaţie cu **WIDTH 1**.

*periferic* este o expresie şir pentru identificatorul perifericului. Perifericele valide sînt SCRN:, LPT1:, LPT2:, LPT3:, COM1: sau COM2:.

*nrfis* este o expresie numerică cu valori între 0 şi 15. Este numărul fişierului deschis ca periferic de ieşire.

În funcţie de perifericul specificat, sînt posibile următoarele acţiuni:

**WIDTH mărime sau WIDTH "SCRN:",mărime**

Fixează lăţimea ecranului. Pentru *mărime* sînt permise doar valorile 40 sau 80. **WIDTH 40** nu este permis pentru adaptorul monocrom.

Specificînd lăţimea la 255 se consideră lăţime "infinită". Această lăţime este utilizată de obicei cu "COM1:" sau "COM2:" pentru fişierele de comunicaţii.

**Exemple:**

**WIDTH "LPT1:",75**

fixează lăţimea de 75 caractere la imprimantă.

**WIDTH 40**

fixează lăţimea ecranului la 40 caractere şi este acelaşi cu:

**WIDTH "SCRN:",40**

## WINDOW

Instrucţiune

**Scop:** Redefineşte coordonatele ferestrei. Are semnificaţie doar în modul grafic.

**Format:** *WINDOW [ [SCREEN](x1,y1)-(x2,y2)]*

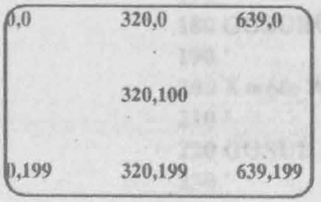
**Comentariu:** *(x1,y1),(x2-y2)* sînt coordonatele definite de programator şi se numesc coordonate universale. Aceste coordonate sînt numere reale în simplă precizie. Ele definesc spaţiul coordonatelor universale ce va fi transformat în spaţiu de coordonate fizice cu ajutorul instrucţiunii VIEW.

Instrucţiunea **WINDOW** permite desenarea unor obiecte (in coordonate univerale) ce nu sînt limitate de marginile ecranului (coordonate fizice). Aceasta se face specificînd perechea de coordonate universale *(x1,y1)* si *(x2,y2)*. Limbajul converteşte, apoi, perechea de coordonate universale pentru a putea fi plotată în fereastră. Pentru a putea face această transformare din spaţiul universal în spaţiul fizic al ecranului, limbajul trebuie să cunoască care porţiune a spaţiului universal nemărginit conţine informaţia de plotat. Această regiune rectangulară în spaţiul coordonatelor universale se numeşte *fereastră*.

În sistemul de coordonate fizice, dacă se rulează:

**NEW**
**SCREEN 2**

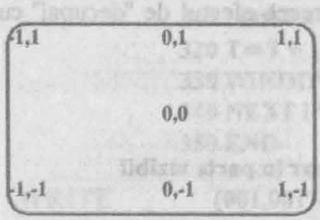
ecranul apare în coordonatele standard:




Dacă se omite atributul **SCREEN**, ecranul este văzut în coordonate carteziene. De exemplu introducînd:

**WINDOW (-1,-1)-(1,1)**

ecranul va apare:

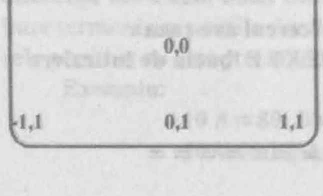



De notat faptul că coordonata y este inversată, astfel (x1,y1) este coordonata jos-stînga, iar (x2,y2) este coordonata dreapta-sus.

Dacă se include atributul **SCREEN**, coordonatele nu sînt inversate, astfel (x1,y1) este coordonata sus-stînga, iar (x2,y2) este coordonata jos-dreapta. De exemplu:

**WINDOW SCREEN (-1,-1)-(1,1)**

definește ecranul:



Este important de notat faptul că instrucţiunea **WINDOW** sortează perechile de argumente x şi y, plasînd primele valorile mai mici pentru x şi y. De exemplu:

**WINDOW (100,100)-(5,5)**

devine:

**WINDOW (5,5)-(100,100)**

Un alt exemplu:

**WINDOW (-4,4)-(4,-4)**

devine:

**WINDOW (-4,-4)-(4,4)**

Orice pereche de x şi y este validă cu singura restricţie că x1 nu poate fi egal cu x2 şi y1 nu poate fi egal cu y2.

Instrucţiunea **WINDOW** permite crearea efectelor de "lupă" şi "decupaj".

Dacă nu se specifică nici un argument, se trece din nou la coordonatele fizice.

**Exemple:** Exemplul de mai jos ilustrează efectul de "decupaj" cu ajutorul instrucţiunii **WINDOW**:

```
10 SCREEN 2: CLS
20 WINDOW (-6,-6)-(6,6)
30 CIRCLE (4,4),5,1
40 ' cercul este mare si doar în parte vizibil
50 WINDOW (-100,-100)-(100,100)
60 CIRCLE (4,4),5,1 ' cercul este foarte mic
70 END
```

Exemplul următor ilustrează efectul de "zoom":

```
10 KEY OFF: CLS: SCREEN 1,0
20 '
30 GOTO 160
40'= = = = = = = = = = = = = = = = = =
50 ' procedura afisaj
60 '
70 LINE (X,0)-(-X,0),,,&HAA00 'axa x
80 LINE (0,X)-(0,-X),,,&HAA00 'axa y
90 '
100 CIRCLE (X/2,X/2),R 'cercul are raza r
110 FOR P=1 TO 500: NEXT P 'bucla de intirziere
120 '
130 RETURN
140'= = = = = = = = = = = = = = = = = =
```

```
150 '
160 X=1000: WINDOW(-X,-X)-(X,X): R=20
170 '
180 GOSUB 50: FOR P=1 TO 1000: NEXT P: CLS
190 '
200 X=60: WINDOW (-X,-X)-(X,X): R=20
210 '
220 GOSUB 50: FOR P=1 TO 1000: NEXT P: CLS
230 '
240 X=100: WINDOW (-5,-5)-(X,X): R=20
250 '
260 GOSUB 50: FOR P=1 TO 1000: NEXT P: CLS
270 '
280 PRINT ".... un exemplu": PRINT ". de zooming.."
290 FOR P=1 TO 2000: NEXT P
300 CLS: T=-50: U=100: X=U: R=20
310 FOR I=1 TO 45
320 T=T+1: U=U-1: X=X-1
330 WINDOW (T,T)-(U,U): CLS: GOSUB 50
340 NEXT I
350 END
```

## WRITE

Instrucţiune

**Scop:** Trimite date spre ecran.

**Format:** *WRITE [lista de expresii]*

**Comentariu:** *lista de expresii*

este o listă de expresii şir şi/sau numerice, separate prin punct şi virgulă sau virgule.

Dacă lipseşte lista de expresii, se afişează o linie liberă.

La afişare, valorile expresiilor numerice sînt separate prin virgule, iar expresiile şir sînt delimitate de ghilimele. După ultimul termen al listei de expresii limbajul adaugă un "carriage return/line feed".

Instrucţiunea **WRITE** este similară cu instrucţiunea PRINT, singura diferenţă între cele două instrucţiuni fiind că **WRITE** inserează virgule între termeni, iar şirurile sînt delimitate de ghilimele. De asemenea numerele pozitive nu sînt precedate de spaţii.

**Exemplu:**

```
10 A=80: B=90: C$="ABCD"
20 WRITE A,B,C$
```

```
RUN
80,90,"ABCD"
```

## WRITE #

Instrucţiune

**Scop:** Scrie date către un fişier secvenţial.

**Format:** *WRITE #nrfis,lista de expresii*

**Comentariu:** *nrfis* este numărul sub care a fost deschis pentru ieşire un fişier.

*lista de expresii* este o listă de expresii şir şi/sau numerice, separate prin punct şi virgulă sau virgule.

La afişare, valorile expresiilor numerice sînt separate prin virgule, iar expresiile şir sînt delimitate de ghilimele. După ultimul termen al listei de expresii limbajul adaugă un "carriage return/line feed". Dacă lipseşte lista de expresii, se scrie o linie liberă.

Instrucţiunea **WRITE #** este similară cu instrucţiunea PRINT #, singura diferenţă între cele două instrucţiuni fiind că **WRITE #** inserează virgule între termeni, iar şirurile sînt delimitate de ghilimele. De asemenea numerele pozitive nu sînt precedate de spaţii.

**Exemplu:** Considerînd că A = 1.23 si B$ = "ABCD", instrucţiunea:

```
WRITE #1,A,B$
```

va scrie următoarea imagine în fişier:

```
1.23,"ABCD"
```

# BIBLIOGRAFIE


*** - IBM BASIC Handbook, General Programming Information, 1984

*** - IBM, BASIC Reference, 1984

**Clarance B. Germain** - Programming the IBM PC & XT, Robert J. Brady Co., 1984

**L.J. Goldstein, M.D. Bowie** - Advanced BASIC and beyound for the IBM PC, Robert J. Brady, 1984

**G. Held** - IBM PC User's reference manual, Hayden Book, 1985

**D. Simon** - IBM BASIC from the ground up, Hayden Book, 1985

**H. Simson** - Serious programming in BASIC, TAB Books INC., 1986

# INDEX

PROGRAME DE INSTRUIRE

Redactor: ION MIHAI
Tehnoredactor: MARILENA DAMASCHINOPOL

---

Coli tipar: 8 $^1/_2$
Bun de tipar: 25.09.1989

---

Tiparul executat sub Cda. 019113 la
Întreprinderea Poligrafică Sibiu
Republica Socialistă România

PROGRAME DE INSTRUIRE

REDACTOR: ION MIHAI
TEHNOREDACTOR: MARILENA DAMASCHINOPOL

---

COLI TIPAR: 8 $^1/_2$
BUN DE TIPAR: 25.09.1989

---

TIPARUL EXECUTAT SUB CDA. 019113 LA
ÎNTREPRINDEREA POLIGRAFICĂ SIBIU
REPUBLICA SOCIALISTĂ ROMÂNIA

EDITURA
ȘTIINȚIFICĂ ȘI ENCICLOPEDICĂ

ISBN 973-29-0108-X

Lei 11,50